NASA-TM-111477

7N 6 [illegible]

Compiling Redesign Plans and Diagnosis Rules from a Structure/Behavior Device Model

Richard M. Keller Catherine Baudin
Yumi Iwasaki Pandurang Nayak Kazuo Tanaka

Technical Report FIA-90-11-07-01
November 1990

NASA Ames Research Center
Artificial Intelligence Research Branch
Mail Stop 244-17
Moffett Field, CA 94035-1000

Compiling Redesign Plans and Diagnosis Rules from a Structure/Behavior Device Model

by

Richard M. Keller*
Catherine Baudin**
NASA Ames Research Center

Yumi Iwasaki
Pandurang Nayak
Knowledge Systems Laboratory, Stanford University

Kazuo Tanaka
NTT Human Interface Laboratories

[To appear as a chapter in the forthcoming book "Knowledge Aided Design", edited by Marc Green, to be published by Academic Press (London), 1991.]

* Employed under contract NAS2-13210 to Sterling Federal Systems.
** Employed under contract NAS2-12952 to Recom Software, Inc.

ABSTRACT

The current generation of expert systems is fueled by special-purpose, task-specific associational rules developed with the aid of domain experts. In many cases, the expert has distilled or compiled these so-called "shallow" rules from "deeper" models of the application domain in order to optimize task performance. With the traditional knowledge engineering approach, only the shallow, special-purpose rules are elicited from the expert -- not the underlying domain models upon which they are based. This results in two significant problems. First, expert systems cannot share knowledge bases because they contain only special-purpose rules and lack the underlying general domain knowledge that applies across tasks. Second, because the underlying models are missing, shallow rules are unsupported and brittle.

This chapter describes a proposed second generation expert system architecture that addresses these problems by linking special-purpose rules to underlying domain models using a process called rule compilation. Rule compilation starts with a detailed domain model, and gradually incorporates various simplifying assumptions and approximations into the model, thereby producing a series of successively less general -- but more task-efficient -- models of the domain. The end product of the rule compilation process is an associational rule model specialized for the task at hand. The process of rule compilation is illustrated with two simple implemented examples. In the first, a structure/behavior model of a simple engineered device is compiled into a set of plans for redesign. In the second, the same underlying device model is compiled into a set of fault localization rules for troubleshooting.

Table of Contents

1. INTRODUCTION AND MOTIVATION

The current generation of expert systems is fueled by special-purpose, task-specific associational rules that have been developed with the aid of domain experts. In many cases, the expert has distilled or "compiled" these so-called "shallow" rules from general domain principles in order to optimize problem solving performance for a specific task context [Anderson 86, Laird et al. 87]. The rules incorporate assumptions that are appropriate to the context, and thereby simplify the necessary reasoning. In other words, compiled expert rules trade generality for efficiency in problem solving.

The task-specific nature of expert rules can be viewed as either a strength or a weakness of current systems, depending on one's perspective. Because the rules are elicited in the context of a specific task (e.g., design, diagnosis), they can be custom-crafted, or optimized, to work efficiently toward the solution of that particular task. The task also serves to circumscribe and focus the knowledge acquisition process. A knowledge engineer need only represent the rules or heuristics necessary to solve one specific task or class of tasks.

Unfortunately, the use of task-specific rules also limits the capabilities of today's expert systems. Despite the intensive effort required to construct rule bases, virtually none of the rules acquired are transferrable to different task situations. A task-specific rule base does not constitute a *reusable* resource -- diagnostic rules, for instance, cannot be used to solve a design problem. Furthermore, the scope of a system's problem solving expertise is severely narrowed by the use of task-specific rules. If a problem situation exhibits characteristics not anticipated by the knowledge engineer, the custom-crafted rules exhibit a type of "brittleness" -- they may apply incorrectly, or fail to apply when appropriate. In contrast, a human expert understands the general domain principles upon which the task-specific rules are based, and can delimit their scope of applicability. The expert can reason from more basic knowledge when simple associational rules are insufficient, bypassing problems of brittleness.

Despite their limitations, we do *not* advocate the elimination of task-specific associational rules from knowledge-based systems. The efficiency benefits associated with "shallow" rules are too great to ignore. Instead, this paper investigates the advantages of a next generation architecture that incorporates *both* general-purpose models of the domain *and* efficient, task-specific rules -- as well as structures that explicitly bridge the gap between the two forms of knowledge.

For example, consider the knowledge-based systems architecture illustrated in Figure 1. This architecture was designed as a next-generation knowledge-based system for reasoning about engineered devices. At the core of the system is a detailed device model, incorporating knowledge of device structure, function, and the physical principles underlying device operation. Associated with the model is a "first principles" reasoning engine that is capable of reasoning about and simulating the behavior of the device under a variety of conditions. In theory, the reasoning engine has the ability to perform diagnosis or redesign from first principles (in the manner of [de Kleer & Brown 84, Davis 84, Genesereth 84]). However, relying solely on first principles reasoning may be infeasible due to problems of computational intractability. To bypass some of the complexity associated with first principles reasoning, the system uses task knowledge in conjunction with a so-called "knowledge compilation" techniques [Dietterich 86] to produce a set of customized, task-specific inference rules. The rules constitute a specialized representation of the general-purpose knowledge found in the device model. The rule representation can be interpreted by a special-purpose inference engine that is optimized for solving a particular type of task (e.g., diagnosis). During the process of rule compilation, the compiler produces bridging structures called justifications. Justifications record the relationship between the compiled rules and the underlying first principles knowledge used in rule compilation.

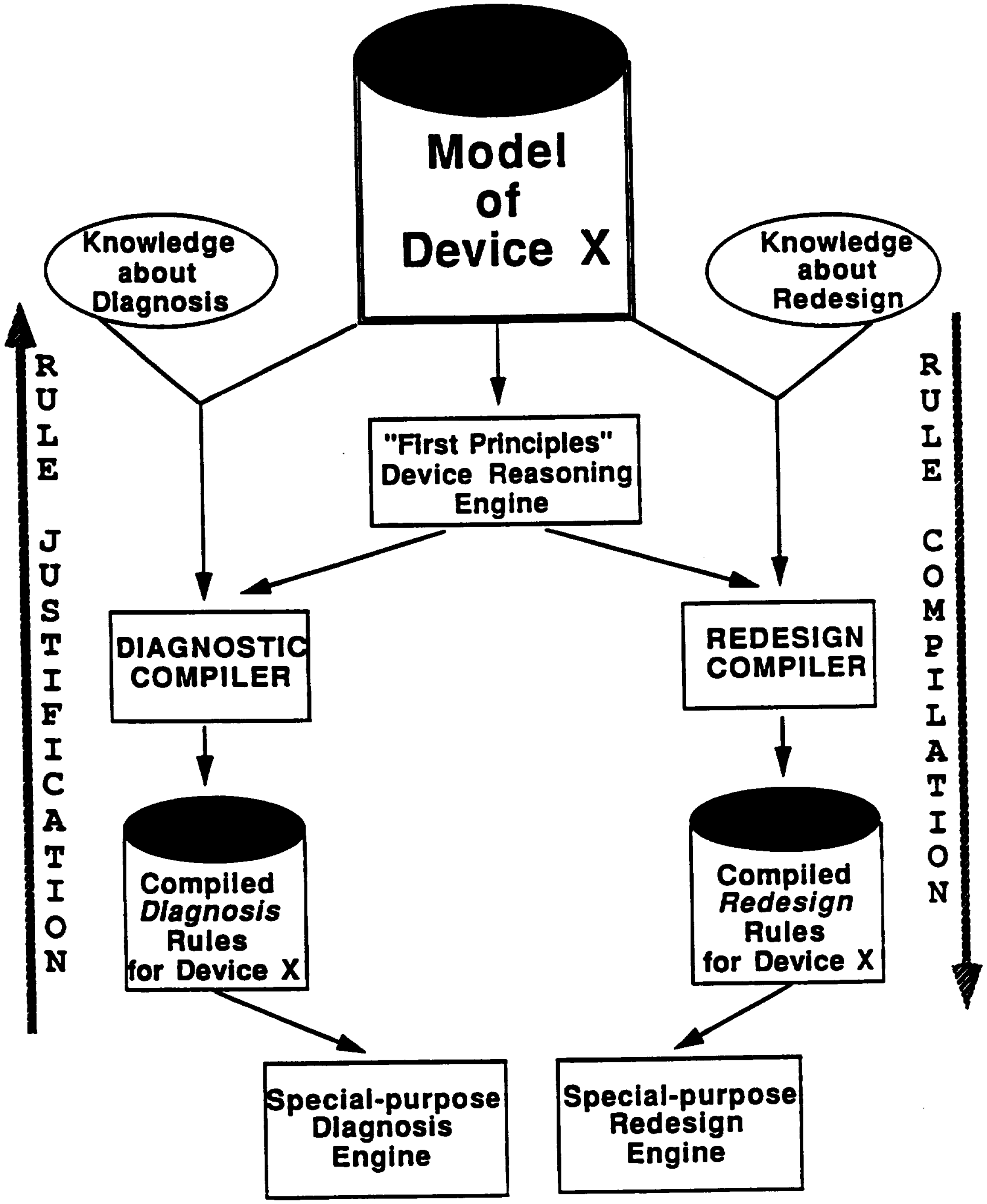

Model of Device X
Knowledge about Diagnosis
Knowledge about Redesign
"First Principles" Device Reasoning Engine
DIAGNOSTIC COMPILER
REDESIGN COMPILER
Compiled *Diagnosis* Rules for Device X
Compiled *Redesign* Rules for Device X
Special-purpose Diagnosis Engine
Special-purpose Redesign Engine
RULE JUSTIFICATION
RULE COMPILATION

Figure 1: An Architecture for Next-Generation Systems

This architecture addresses both the brittleness and non-reusability problems described above. By constructing a general-purpose device model, rather than a special-purpose rule base, the domain knowledge is available to support inferencing across a wider variety of tasks. And through knowledge compilation, the system can transform the device model into an efficient, task specific rule representation in order to avoid the intractability problems associated with first principles inferencing. But in contrast to current systems, when the shallow rules fail, the system can resort to a first principles analysis by examining the justification knowledge recorded with the rules. This architecture synthesizes various approaches, including work on rule justification [Smith et al. 85], "shallow" knowledge compilation [Chandrasekaran & Mittal 83, Sembugamoorthy & Chandrasekaran 86, Araya & Mittal 87, Brown & Sloan 87], explanation [Swartout 83], and operationalization [Mostow 81, Keller 87].

The purpose of this paper is to explore the plausibility of this next generation architecture by presenting the results of a one-year feasibility study that was proposed in [Keller et al. 87] and conducted at the Knowledge Systems Laboratory with the support of NASA's Ames Research Center and Marshall Space Flight Center. The study involved the construction of simple knowledge compilers capable of producing selected task-specific rules (for both a diagnostic and a redesign task) from a general-purpose model of a simple, engineered device -- the Reaction Wheel Assembly of NASA's Hubble Space Telescope. This chapter describes the device representation and the rule compilation mechanism in detail. The rest of the chapter is organized as follows: In the next section, we describe the Reaction Wheel Assembly, and present our general-purpose model of the device. Then, we describe the functioning of two implemented knowledge compilers, one that compiles redesign plans and the other that compiles diagnostic rules. Finally, we analyze the results of the study and discuss its limitations.

2. MODELING THE REACTION WHEEL ASSEMBLY

The Reaction Wheel Assembly (RWA) is part of the pointing and control subsystem aboard NASA's Hubble Space Telescope (HST). This electromechanical device was a reasonable candidate to model because it consists of a small number of components, and functions according to relatively simple physical principles. The RWA served as a useful testbed for our research, but any number of simple engineered devices would have sufficed to investigate research issues described above. This section describes the RWA and presents a simple model of its structure and behavior.

2.1. Description of RWA

The function of the Reaction Wheel Assembly device (RWA) is to point the Space Telescope at the proper area of the sky, and keep the telescope locked onto its target. The RWA does not make use of thruster jets to control HST's position -- exhaust vapors would damage the telescope's sensitive optical instruments. Instead, the RWA functions according to a very simple physical principle -- conservation of angular momentum. The reaction wheel acts like a spinning top within the telescope (Figure 2). When the reaction wheel is at rest, the telescope is stationary, floating in orbit around the earth. When NASA scientists wish to move the barrel of the telescope, they start the reaction wheel spinning. Due to conservation of angular momentum, the telescope starts spinning in the opposite direction from the wheel. When the telescope nears its proper orientation, the spin is reversed and the telescope slows down.[1] There are four reaction wheels aboard HST, and the sum of the torque forces generated by these wheels enables the telescope to rotate about an arbitrary axis.

[1]Actually, this explanation is somewhat oversimplified. The reaction wheel must be moving at all times to counteract the effects of the earth's gravitational field, which cause a small torque even when HST is "stationary".

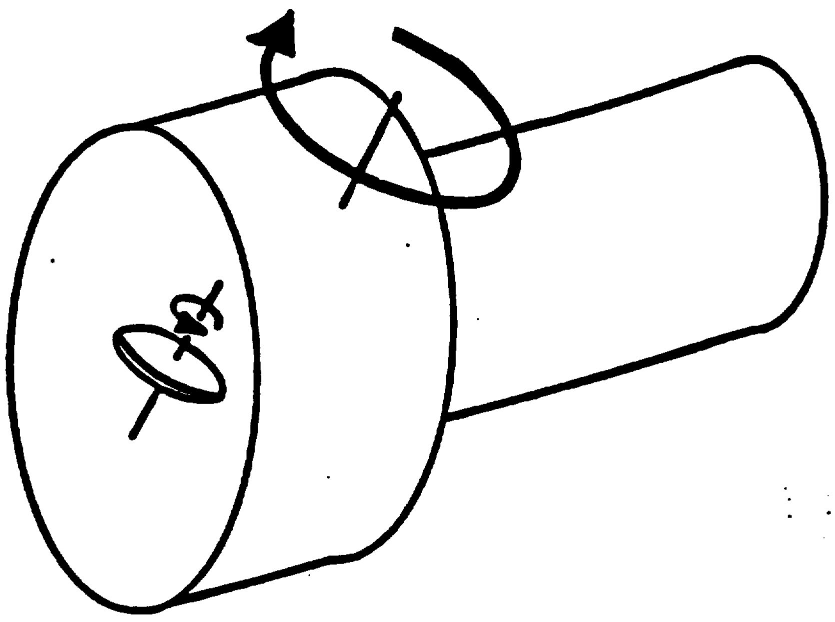


Figure 2: How the RWA functions

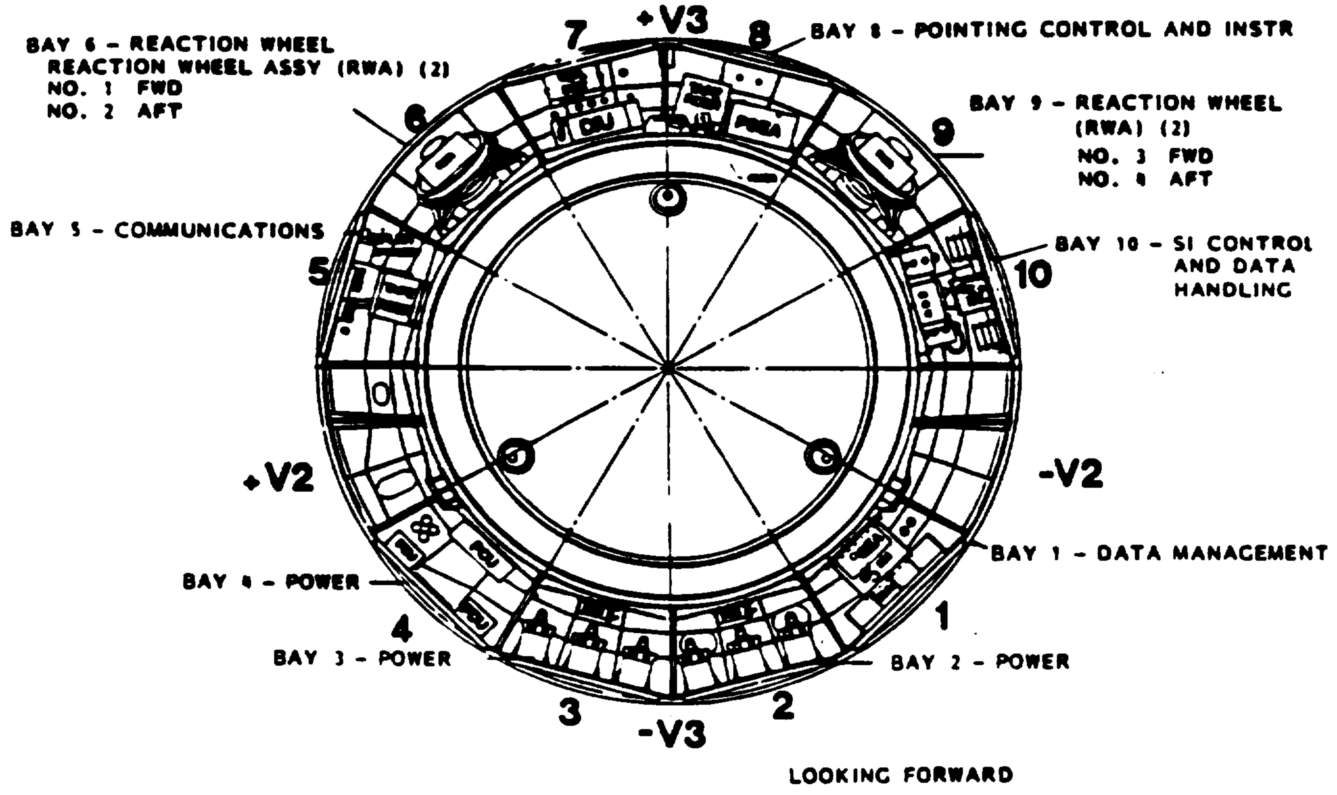


Figure 3: HST transport bays (from [LMSC 84, Figure 1-2], used with permission)

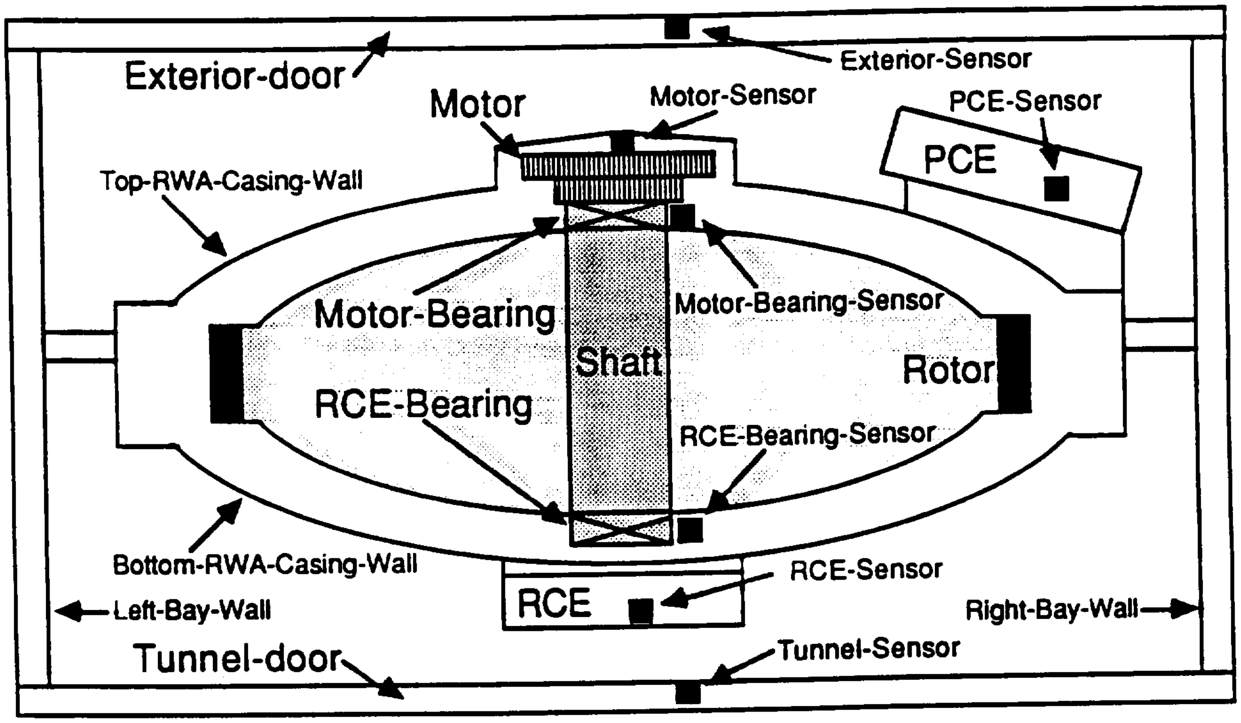


Figure 4: RWA cross-sectional view (adapted from [Perkins & Austin 87], used with permission). The outside shell of the RWA consists of a metal casing, which mounts directly to the telescope bay walls. Inside the casing is a hollow aluminum wheel with a steel rim, mounted on a rotating shaft. The shaft is connected to a motor at the top of the assembly. The Rotor Control Electronics (RCE) and the Power Control Electronics (PCE) supply power and control signals to the motor. Each end of the wheel shaft is supported by a bearing. The top bearing is called the Motor-Bearing, and the bottom bearing is called the RCE-Bearing (due to their proximity to the motor and the RCE, respectively). Near each of the heat-generating components within the RWA (e.g., the bearings) there is a small temperature sensor used to monitor the device's functioning.

The reaction wheels are mounted within the transport bays which ring the barrel of the telescope (Figure 3). One end of the RWA points out toward the bay door (and deep space) while the other end points in toward the telescope tunnel. A cross-sectional view of the RWA is presented in Figure 4.

2.2. Device Model for RWA

This section reviews the device model constructed for the RWA. The model was constructed using a frame-based, object-oriented knowledge representation tool called HyperClass[2]. The model consists of two basic parts: a structural representation, and a behavioral representation. These are discussed in the following two subsections.

2.2.1. Structural representation

The structural part of the device representation consists of information about the component/subcomponent structure of the device, including the physical connectivity of the components and the spatial relationships among the components.

2.2.1.1. Device components

Each device component is represented by a different object (frame) in a HyperClass knowledge base. Components come in two different varieties: either complex or primitive. Complex objects are those whose internal substructures are represented explicitly by a set of more detailed, lower-level subcomponents, whereas primitive components are those represented as black boxes. The component/subcomponent hierarchy for the RWA is presented in Figure 5. The breakdown of complex RWA components into lower-level subcomponents corresponds roughly to the structural units referenced in the RWA specifications [LMSC 84].

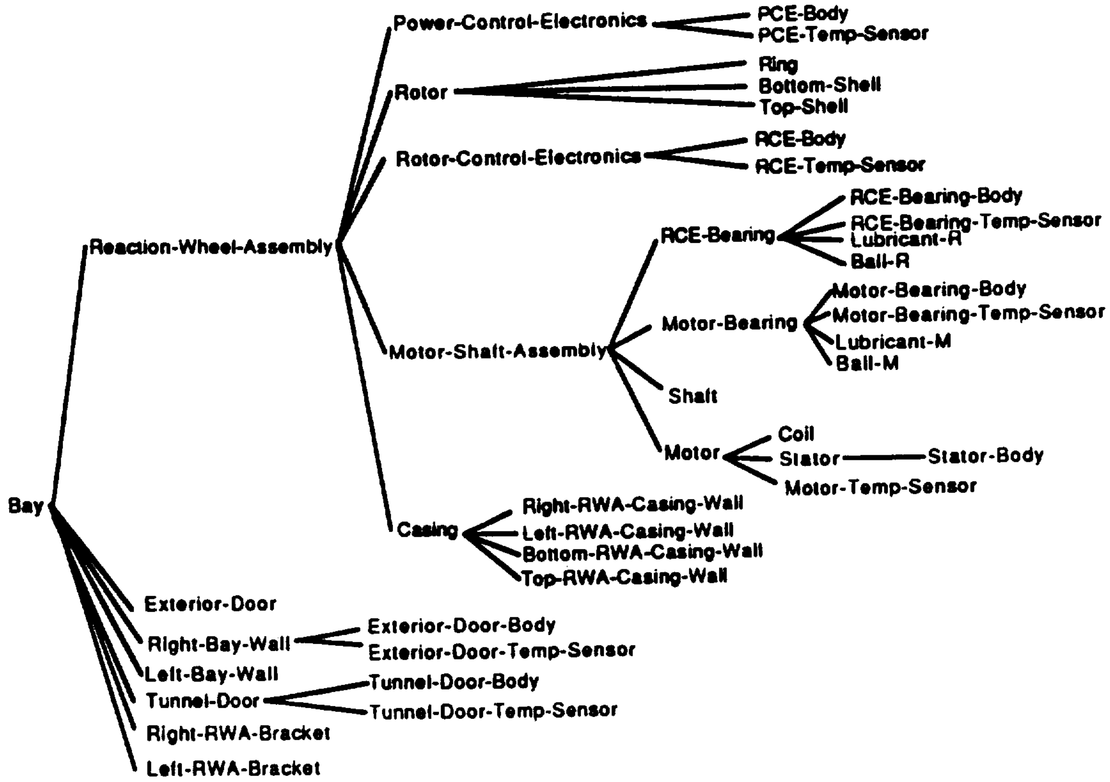


Figure 5: Component/subcomponent hierarchy for RWA

[2]HyperClass is a trademark of Schlumberger Technologies, Inc. HyperClass is a successor of the Strobe knowledge representation tool [Smith 83].

2.2.1.2. Spatial representation

For the purposes of our initial study, we utilized a simple two-dimensional, bounding box spatial representation. Each device component is circumscribed with a rectangle, and the coordinates of the lower left corner of the device are recorded with respect to the reference axis for the device's supercomponent. The bounding box representation for RWA is presented in Figure 6. Components listed in boldface are complex components.

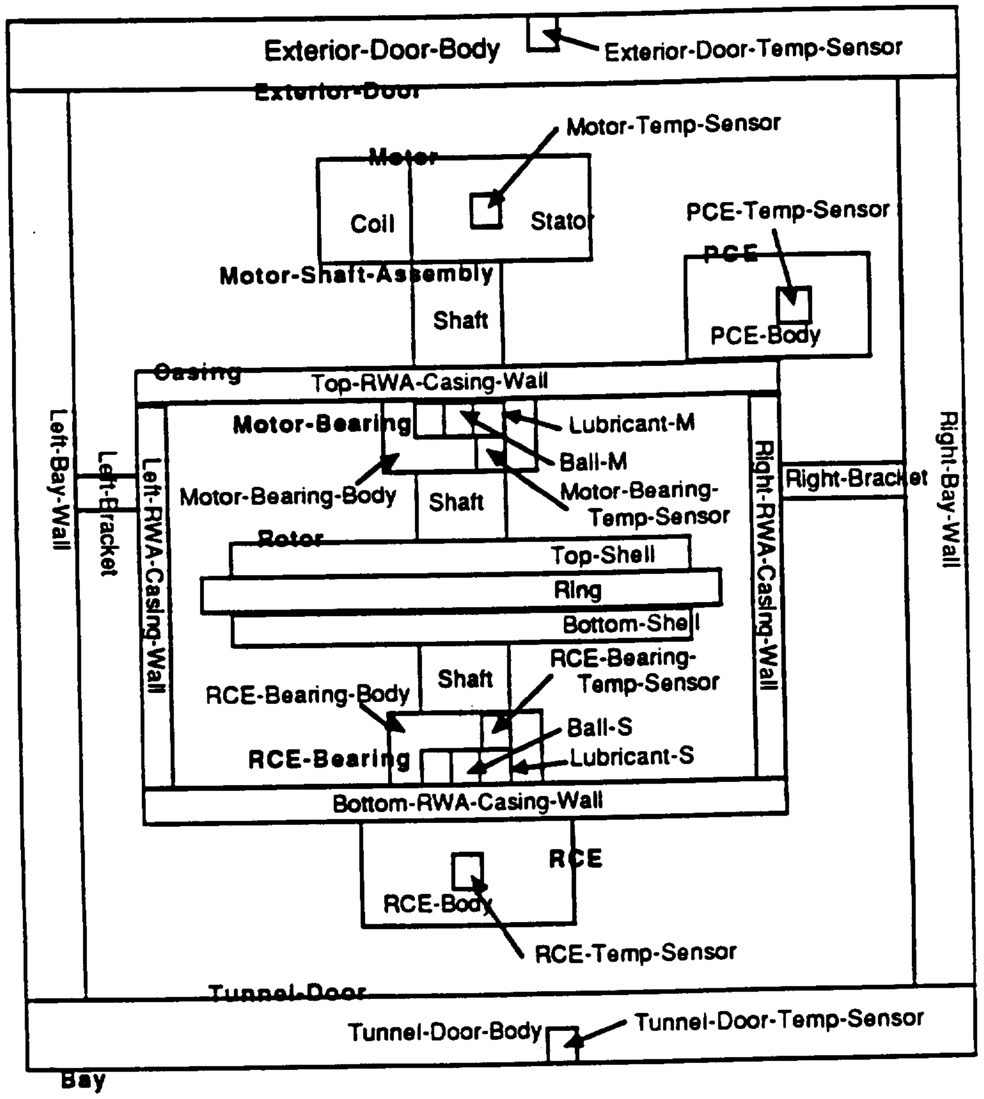


Figure 6: Bounding box representation for RWA

2.2.1.3. Physical Connectivity

Physical connections between components are modeled as connection objects, with each high-level connection being elaborated in terms of lower-level connections. Figure 7 illustrates the connection structure of the subcomponents within the Reaction-Wheel-Assembly component. The children of Reaction-Wheel-Assembly in Figure 7 represent all physical connections among its constituent subcomponents. These connections are elaborated at a greater level of detail as connections between lower-level subcomponents.

For example, the connection between the PCE and the Casing is elaborated as a connection between the body of the PCE and the top casing wall (PCE-Body+T-RWA-C-Wall). In general, the physical connection object provides a place to record information about the nature of the connection between two components (e.g., whether it is a hinge, a weld, a bolt, etc.).

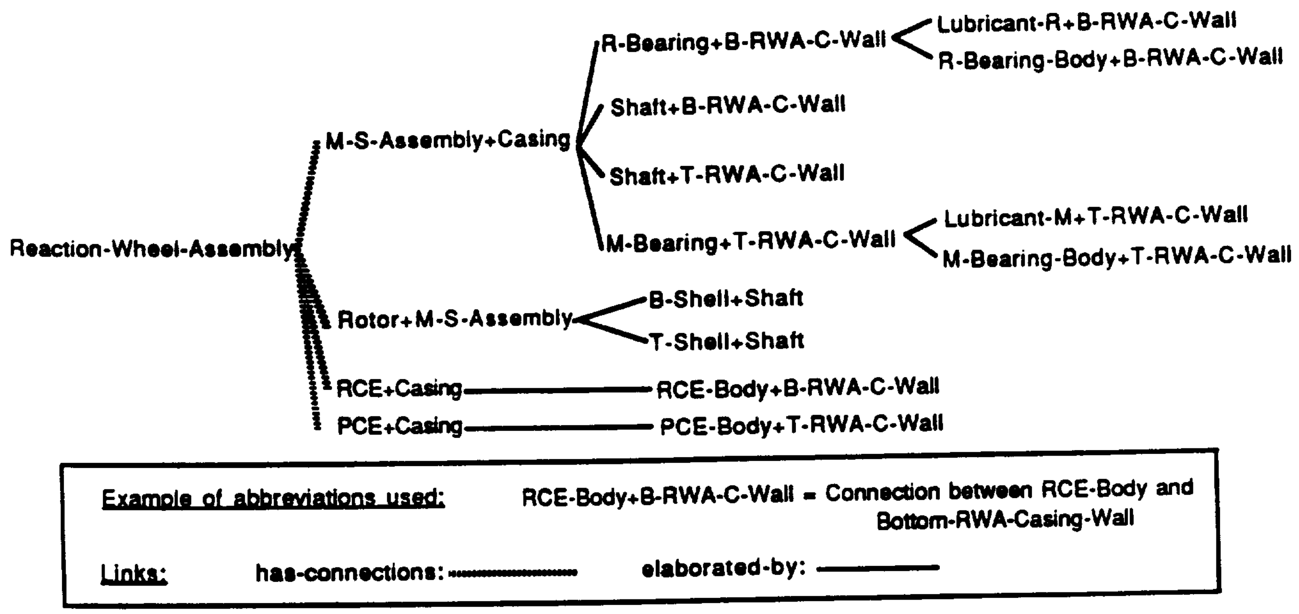


Figure 7: Physical connectivity of RWA components

2.2.1.4. Component materials

Each primitive component is modeled as being composed of a single material. Materials are represented as objects in the knowledge base. Various intrinsic properties associated with a material – such as coefficient of reflectivity and density – are stored in the material object.

2.2.1.5. Example

To clarify the representation of device structure, following is an instance of the HyperClass object that represents the RWA component. Object names are printed in uppercase.

Object: REACTION-WHEEL-ASSEMBLY-1 (RWA-1)
Type: individual
Generalizations: ACTUATOR ; a type of COMPONENT
Specializations: nil
*Component*Inv: BAY-6*
Width: 24.0
Height: 22.0
*Subcomponents: (PCE*s RCE*s Rotor*s Casing*s Motor-shaft-assembly*s)*
*Connections: (M-s-assembly+casing*s PCE+casing*s Rotor+m-s-assembly*s RCE+casing*s)*

```
PCE*s: PCE-22
            X-position: 18.0
            Y-position: 16.0
RCE*s: RCE-17
            X-position: 7.0
            Y-position: 0.0
Rotor*s: ROTOR-8
            X-position: 2.0
            Y-position: 8.0
Casing*s: CASING-66
            X-position: 0.0
            Y-position: 3.0
Motor-shaft-assembly*s: MOTOR-SHAFT-ASSEMBLY-2
            X-position: 6.0
            Y-position: 4.0
M-s-assembly+casing*s: MOTOR-SHAFT-ASSEMBLY+CASING-36
PCE+casing*s: PCE+CASING-12
Rotor+m-s-assembly*s: ROTOR+M-S-ASSEMBLY-16
RCE+casing*s: RCE+CASING-32
```

The RWA is a kind of ACTUATOR component, and inherits slots from this class of objects. The supercomponent of RWA-1 is BAY-6, listed in the Component*Inv (component-inverse) slot. Conceptually, therefore, the RWA is considered a subcomponent of the Space Telescope bay in which it is physically located. The Width and Height slots give the dimensions of the bounding box for RWA-1. (Measurement units are unspecified.) The lower left corner coordinates of the RWA-1 bounding box are recorded in its supercomponent, BAY-6. The Subcomponents slot of RWA-1 contains a list of slots that point to each of its subcomponents.[3] These slots (e.g., PCE*s) establish the lower left corner coordinates of each subcomponent's bounding box relative to RWA-1's lower left corner. The Connections slot contains a list of slots that describe the physical connections between the RWA-1's components. For example, the connection between the Casing and the RCE is represented by the contents of the slot PCE+casing*s, which consists of a PHYSICAL-CONNECTION object: PCE+CASING-12.

```
Object: PCE+CASING-12
Type: individual
Generalizations: PHYSICAL-CONNECTION
Specializations: nil
Compl: PCE-22
            Object: RWA
            Slot: PCE*s
Comp2: CASING-66
            Object: RWA
            Slot: Casing*s
Elaborations: (PCE-body+t-RWA-c-wall*s)
PCE-body+t-RWA-c-wall*s: PCE-BODY+T-RWA-C-WALL-12
```

PCE+CASING-12 is a binary connection between the components pointed to by the Compl and Comp2 slots. These slots also record backpointers to the component's supercomponent and the slot name in which the component appears. The Elaborations slot contains a list of slots which point to a more detailed description

[3]A representational alternative would be to avoid the indirect reference through slots by recording the subcomponent objects directly in the Subcomponents slot. However, the indirect referencing scheme permits us to associate additional meta-level information with the component descriptions (e.g., bounding box coordinates). This representation also helps us to declaratively encode the structure of generic components, irrespective of how they are instantiated.

of the physical connection. More specifically, PCE-22 and CASING-66 are actually connected because PCE-BODY-22 touches TOP-RWA-CASING-WALL-3. This fact is reflected in the following object:

```
Object:  PCE-BODY+T-RWA-C-WALL-12
Type:  individual
Generalizations:  PHYSICAL-CONNECTION
Elaboration*inv:  PCE+CASING-12
Elaborations: nil
Comp1:  PCE-BODY-22
            Object:  PCE
            Slot:  PCE-body*s
Comp2:  TOP-RWA-CASING-WALL-3
            Object:  CASING
            Slot:  Top-RWA-casing-wall*s
```

Notice that this representation permits connections between components that are not immediate subcomponents of the same supercomponent.

2.2.2. Behavioral representation

Device behavior is represented by a set of behavioral equations. These equations specify constraints among numeric quantities associated with device components. The behavioral equations are represented as ordinary quantitative equations as well as qualitative differential equations to facilitate different types of reasoning. Quantitative equations can be used to compute exact values for quantities, whereas qualitative differential equations can be used to determine how changes in quantities propagate through a device. Quantitative equations represent precise numerical relationships between quantities (e.g., $F=ma$). Qualitative differential equations, on the other hand, represent much weaker relationships between changes in quantities (e.g., the qualitative differential equation corresponding to $F=ma$ says, among other things, that if m or a decrease, F must also decrease).

2.2.2.1. Quantities

Each numeric quantity or parameter associated with a component (e.g., Mass, Velocity) is represented by a separate slot in that component's description. The quantity slot points to an instance of a QUANTITY object in the knowledge base. The QUANTITY object bundles together all information pertaining to the quantity, including its value, procedures for computing the value, simple constraints on the value, and any working assumptions about the sign and/or magnitude of the quantity. The QUANTITY object also contains a slot called "possible-equations", which provides a pointer to all quantitative equations in which that quantity appears. An example will clarify the relationships among these different types of objects.

Here is a description of the rolling ball component of a ball-bearing in the RWA:

```
Object: BALL-151
Type:  individual
Generalizations: BALL                          ; A type of COMPONENT
Specializations:  nil
Component*inv:  RCE-BEARING-221
Width: 1
Height: 1
Subcomponents: nil
Radius: BALL-RADIUS-102
```

The Radius slot contains BALL-RADIUS-102, which represents a quantity corresponding to the radius of the rolling ball. BALL-RADIUS-102 is described below:

```
Object: BALL-RADIUS-102
Type: individual
Generalizations: BALL-RADIUS                    ; A specialization of QUANTITY
Specializations: nil
Constraints: constant
Possible-equations: (Ball-radius-eqn*s Bearing-friction-eqn*s)
Ball-radius-eqn*s: BALL-RADIUS-EQUATION-32
Bearing-friction-eqn*s: BEARING-FRICTION-EQUATION-103
```

One can see from the above description that BALL-RADIUS-102 appears in two equations---BALL-RADIUS-EQUATION-32 and BEARING-FRICTION-EQUATION-103. The "constraints" slot is valid for numeric quantities, and contains information about constraints imposed on the quantity. Currently, the system can represent three simple types of unary constraints on quantities: keep quantity constant, maximize quantity, and minimize quantity. Such constraints may originate with the design specification documents.

2.2.2.2. Quantitative Equations

Quantitative equations are represented by QUANTITATIVE-EQUATION objects. Each such object specifies an equation in terms of local variables, and provides a mechanism to bind these local variables to actual quantities. QUANTITATIVE-EQUATION objects also have a precondition slot, representing an expression that must be true for the equation to hold. Each QUANTITATIVE-EQUATION object may also be linked to 0 or more qualitative differential equations (represented by QUALITATIVE-EQUATION objects). These qualitative equations represent different versions of the quantitative equation corresponding to different assumptions about the signs of the quantities.

Consider, for example, the following quantitative equation, representing a constraint between the ball's radius and the width of the bearing:

```
Object: BALL-RADIUS-EQUATION-32
Type: individual
Generalizations: BALL-RADIUS-EQUATION           ; A type of QUANTITATIVE-
                                                ; EQUATION
Specializations: nil
Precondition: T                                 ; This equation always holds
Equation: (Equal Radius*s (Multiply 0.5 Width*s))
Locals: (Radius*s Width*s)                      ; List of local variables
Radius*s: (Radius Ball*s)                       ; A path specification to compute
                                                ; bindings
Width*s: (Bearing-width RCE-bearing*s)
Objects: (Ball*s RCE-bearing*s)                 ; Objects with participating
                                                ; parameters
Ball*s: BALL-151
RCE-bearing*s: RCE-BEARING-12
Qualitative-equations: (Qual-ball-radius-eqn*s) ; List of qualitative equations
Qual-ball-radius-eqn*s: QUAL-BALL-RADIUS-EQN-73
```

The bindings for the local variables found in the Locals slot are specified using a path specification. Thus, in the above example, the path specification for Radius*s is (Radius Ball*s), which expands to the Radius slot of the Ball*s slot of the current equation object (i.e., the Radius slot of BALL-151). From the Qualitative-equations slot we see that BALL-RADIUS-EQUATION-32 is linked to qualitative differential equation QUAL-BALL-RADIUS-EQN-73 (see below).

The equation language used to express quantitative constraints is limited to expressing equalities between terms, where terms consist of numbers, local variables, and functions of other terms. Many common arithmetic functions are predefined in the system, and it is easy to add new ones in a modular fashion.

2.2.2.3. Qualitative Equations

Equations are also represented in their qualitative differential form since the qualitative differential form is more convenient than the quantitative form for the purpose of inferring the influence of a change in one variable on others. An ordinary, quantitative equation is converted into a qualitative differential equation by, first, time-differentiating both sides of the equation, and, then, converting the result into a qualitative equation. Qualitative equations express constraints on the signs (+, 0, or -) of variables. In a qualitative equation, variables represent signs of quantities but not their magnitudes. An equation is converted into a qualitative form by discarding constant coefficients except for their signs and replacing the variables by their qualitative versions.

Take, for example, the following equation (1) with variables x, y, and positive constants a and c.

$$x + cy = a. \tag{1}$$

Equation (1) is time-differentiated to produce

$$x' + cy' = 0,$$

where the primed variables (x', y') denote the time derivatives. The qualitative version of this equation is

$$[x'] + [y'] = 0, \tag{2}$$

where $[x']$ and $[y']$ denote the signs of the time-derivatives of x and y, respectively, and the operators are qualitative operators over signs.[4] Qualitative equation (2) expresses the constraint that the signs of x' and y' must be different or they should both be zero. When variables in a qualitative equation are time derivatives as in this case, qualitative equations represent constraints on how the other variable values can change when one of them changes. Thus, according to equation (2), when x increases, y must decrease or vice versa, and when one of them remains constant, the other must also.

A qualitative differential version of the above equation BALL-RADIUS-EQUATION-32 is represented by the QUAL-BALL-RADIUS-EQN-73 object:

```
Object:  QUAL-BALL-RADIUS-EQN-73
Type:  individual
Generalizations:  QUAL-BALL-RADIUS-EQN-73          ; Specialization of Qualitative-
                                                   ; Equation
Specializations:  nil
Equation: ((+ Radius*s) (- Width*s))               ; [Radius*s'] - [Width*s'] = 0
Owner:  BALL-RADIUS-EQUATION-32
Assumptions:  nil
```

[4]See any discussion of qualitative physics (such as [de Kleer & Brown 84] or [Iwasaki 89]) for a more detailed description of qualitative arithmetic.

The Equation slot is filled by a list of pairs, (s_i q_i), where the s_i are either + or -, and the q_i are quantities, specified by local variables. (The local variables used in the Equation slot are bound in the corresponding *quantitative* equation object, which is stored in the Owner slot.) The pairs are interpreted as signed terms in a normalized qualitative differential equation: $s_1\ q_1\ s_2\ q_2 \cdots s_n\ q_n = 0$. The qualitative differential equation ((+ Radius*s)(- Width*s)), for example, specifies that the sign (+, -, 0) of the time derivative of the ball bearing radius minus the sign of the time derivative of the bearing width must equal zero. From this, we can determine that as the ball radius increases, the bearing width must decrease. The Assumptions slot specifies any assumptions that were made in converting the original quantitative equation into a qualitative form. For example, it might be necessary to assume that a particular quantity is non-negative to produce a qualitative form.

3. COMPILING ABSTRACT REDESIGN PLANS

This section describes how the general-purpose model of the RWA presented in Section 2.2 can be compiled into a set of abstract plans for redesign. Here is a sample illustrating the type of abstract redesign plan generated by the compiler:

If *the goal is to decrease the temperature of RCE-Bearing*
then *consider the following actions, in the following order:*
increase the bearing-width of RCE-Bearing;
increase the thickness of Bottom-RWA-Casing-Wall;
increase the thermal-constant of Bottom-RWA-Casing-Wall;
increase the body-width of RCE-body;
increase the thermal-constant of RCE-body

Notice that this plan is abstract in the sense that it does not specify exactly *how* to achieve the desired decrease in temperature, but merely narrows the set of potential actions for achieving the goal down to a small ordered set of recommended actions. Furthermore, the actions do not suggest exact quantitative values for increasing and decreasing the specified quantities. This type of plan serves as a starting point for a redesign system, and must be elaborated further to produce a complete redesign plan in the form of a set of executable actions. For example, the redesign shell developed for the FRM financial planning application [Gelman et al. 88] can accept an abstract redesign plan as input and produce a completely specified final plan as output. The final plan specifies which of the recommended redesign actions should be executed and exactly how much the quantities should be modified. The shell uses the input plan plus knowledge about current values of quantities and constraints on those values to produce the final plan.

The process by which an abstract redesign plan is compiled resembles macro-formation. The plan is extracted from a tree of redesign goals generated during the compilation process. The root of the redesign goal tree constitutes the "if" part of the plan and the leaves of the goal tree, suitably ordered, form the "then" part. The goal tree used in compiling the redesign plan shown above is illustrated in Figure 8.

The following sections detail the series of compilation steps used to transform the general-purpose RWA device model (Section 2.2), first into a *quantitative* behavior model, then into a *qualitative* behavior model, and finally into a *causal* behavior model, before producing the redesign goal tree pictured in Figure 8.

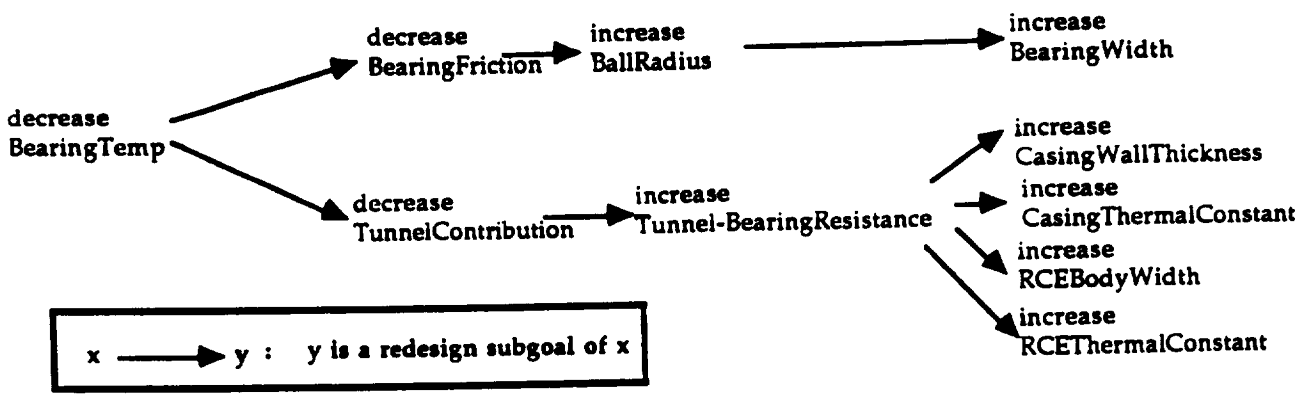

decrease BearingTemp
decrease BearingFriction
increase BallRadius
increase BearingWidth
decrease TunnelContribution
increase Tunnel-BearingResistance
increase CasingWallThickness
increase CasingThermalConstant
increase RCEBodyWidth
increase RCEThermalConstant
x → y : y is a redesign subgoal of x

Figure 8: Final redesign goal tree

3.1. Equation set assembly

The first step in the process of compiling redesign heuristics is to assemble a set of qualitative differential equations, which can be viewed as a qualitative model of the RWA's operation. The qualitative model abstracts away precise numeric constraints and retains only imprecise knowledge of functional relationships that describe changes in quantities. For redesign, this model will be used to draw inferences about how to modify the values of controllable quantities in order to achieve a given redesign goal.

In assembling a qualitative equation model, it is computationally advantageous to limit the scope of the model to the particular subsystem undergoing redesign. The idea is to effectively circumscribe a focal set of quantities (and equations) within the potentially large set of quantities/equations associated with the entire device. So to form an equation model, the compiler uses structural subcomponent links present in the RWA device model to assemble all components within a given, scoping supercomponent (e.g., the BAY in which the RWA sits). Then the compiler gathers together the set of *quantitative* equations that interrelate the quantities associated with those components. (As described in Section 2.2.2.1, each QUANTITY object includes a pointer to the set of quantitative equations in which that quantity appears.) Each of these quantitative equations can be associated in the device model with a set of corresponding qualitative differential equations. In general, there is no unique qualitative differential equation corresponding to a given quantitative equation. To select a single qualitative form, the compiler must make use of assumptions about the signs of the quantities involved. For example, the following quantitative equation can be transformed into any one of three qualitative differential equations, depending on whether the constant, c, is positive, negative, or zero:

Original quantitative equation: $x + cy = z$
After time differentiating: $x' + cy' = z'$
Converting to qualitative format:
case 1: $c = 0$ $[x'] - [z'] = 0$
case 2: $c > 0$ $[x'] + [y'] - [z'] = 0$
case 3: $c < 0$ $[x'] - [y'] - [z'] = 0$

Notation: $[x']$ stands for the sign of the time derivative of x

Given a set of sign assumptions, it is possible in principle to automate the conversion from quantitative to qualitative differential equations. However, in the current version of the system, qualitative differential equations are generated manually from the quantitative equations. Any assumptions about the sign of quantities are recorded in the qualitative equation object (see Section 2.2.2.2). A complete listing of all qualitative equations used in the RWA redesign problem can be found in Appendix A.

3.2. Causal dependency analysis

The next step is to infer causal relationships among the quantities represented in the qualitative equation model. Whereas equations are inherently acausal, the redesign task requires knowledge of causal dependencies to determine how design changes modifying specific quantities propagate through the device to other quantities via equations. Simon & Iwasaki's causal ordering procedure [Iwasaki & Simon 86] is used to analyze causal dependencies and produce a graph structure that encodes these dependencies. In order to apply the procedure, one must have the same number of equations as the variables. Furthermore, the equations must be independent and each of them must represent a conceptually distinct mechanism in the situation. If some of the quantities are known to be exogenous, or quantities that are directly controlled by external factors, an equation of the form $v = c$, where v is a quantity and c is some constant, for each such quantity v, must be included in the set to represent this fact.

Given a set of N equations which satisfy these requirements, the first step of the causal ordering procedure is to isolate all the subsets of quantities whose values can be determined independently of the remaining quantities. Such a subset of quantities can be found by identifying a set of n equations which contains exactly n quantities but which itself does not include a proper subset containing the same number of equations as quantities. Such subset is called a ***minimal complete subset***. The quantities in any minimal complete subset are the "uncaused causes" of the system, and they are causally independent of other quantities. Each exogenous quantity equation consists one minimal complete subset. Next, the equations in all minimal complete subsets are removed from the original set of equations and their quantities are also removed from the remaining equations, producing a reduced set of $N - m$ equations in $N - m$ quantities, where m is the total number of equations (and variables) in all the minimal complete subsets. Then, a new independent subset of quantities is determined in the reduced set. This process repeats until the set can no longer be reduced. For each equation in the original set, the quantity that was reduced last is said to be ***causally dependent upon*** all the other quantities in the equation, and a directed graph can be generated to depict the causal dependency structure of the entire set, with nodes representing quantities and links representing causal dependency relations among them. The exogenous quantities serve as initial, anchoring nodes in the graph. The causal dependency graph computed for the RWA redesign problem is given in Figure 9.

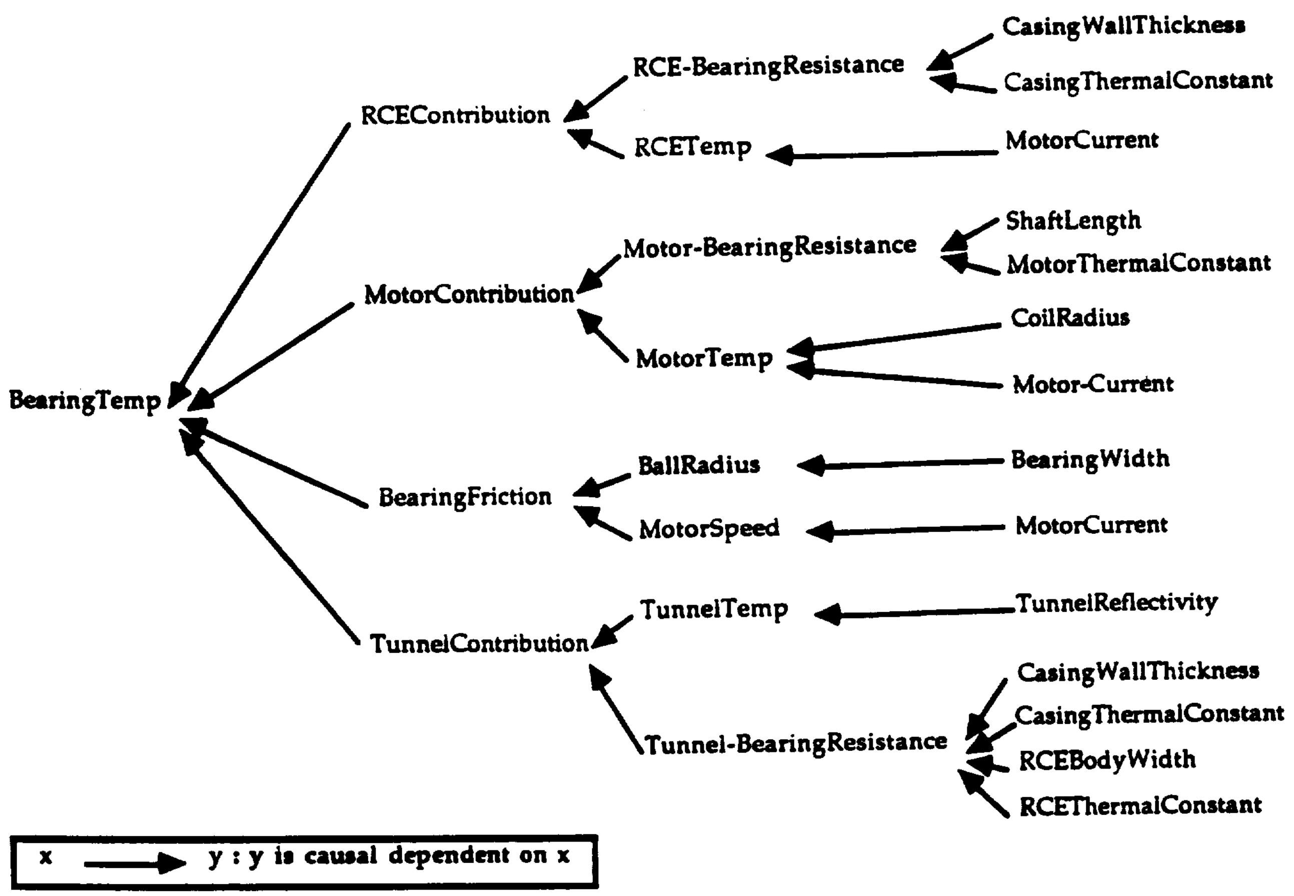


Figure 9: Causal dependency graph derived from Appendix A equations

3.3. Redesign goal tree generation

Once the causal dependency graph is available, the compiler can use the graph plus the qualitative differential equation model to produce a tree of redesign alternatives for achieving a specified redesign goal. Goals take the form **"{increase or decrease} quantity *Q* of component *C*"**. The compiler augments each causal dependency node in the causal subgraph rooted at Q with the action "increase" or "decrease", as appropriate, to convert the causal dependency nodes into redesign subgoals. For any given goal, the qualitative equations can be used to determine the appropriate subgoals to form. For example, suppose the goal is to increase the quantity x, and x is causally dependent on quantities y and z. Further assume that the qualitative differential equation relating x to y and z is $[x'] + [y'] + [z'] = 0$. In order for x to increase, either y must decrease (assuming z constant) or z must decrease (assuming y constant). This generates two corresponding subgoals in the redesign goal tree. The leaves of the goal tree correspond to a set of "increase" or "decrease" actions on externally-controllable (exogenous) quantities, and thus constitute an unordered set of executable actions for achieving the top-level goal. The initial redesign goal tree generated in this manner for the problem of decreasing the temperature of the RCE bearing (BearingTemp) is presented in Figure 10. The leaves of the goal tree correspond to a set of actions on controllable (exogenous) quantities, and thus constitute an unordered set of executable actions for achieving the top-level goal.

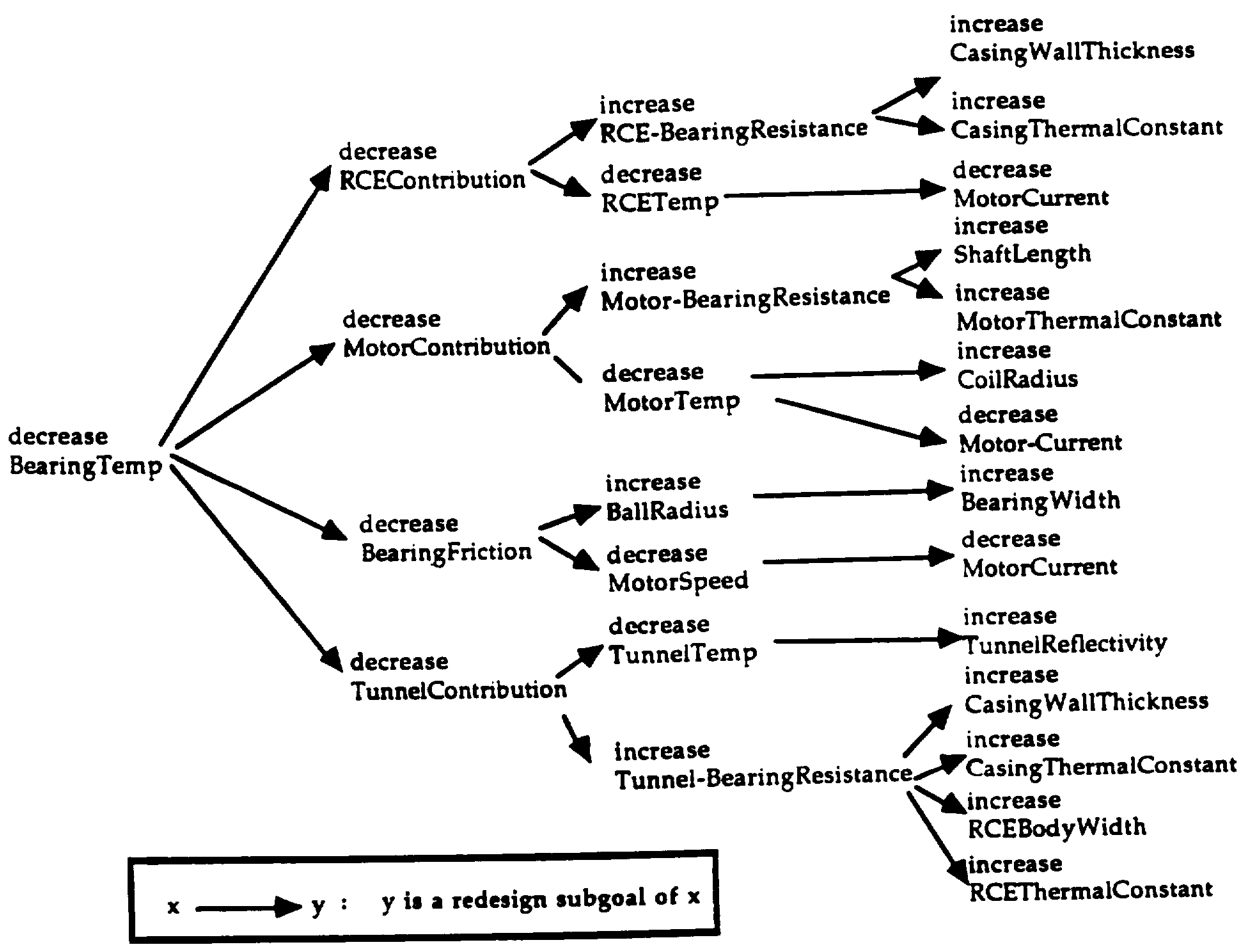
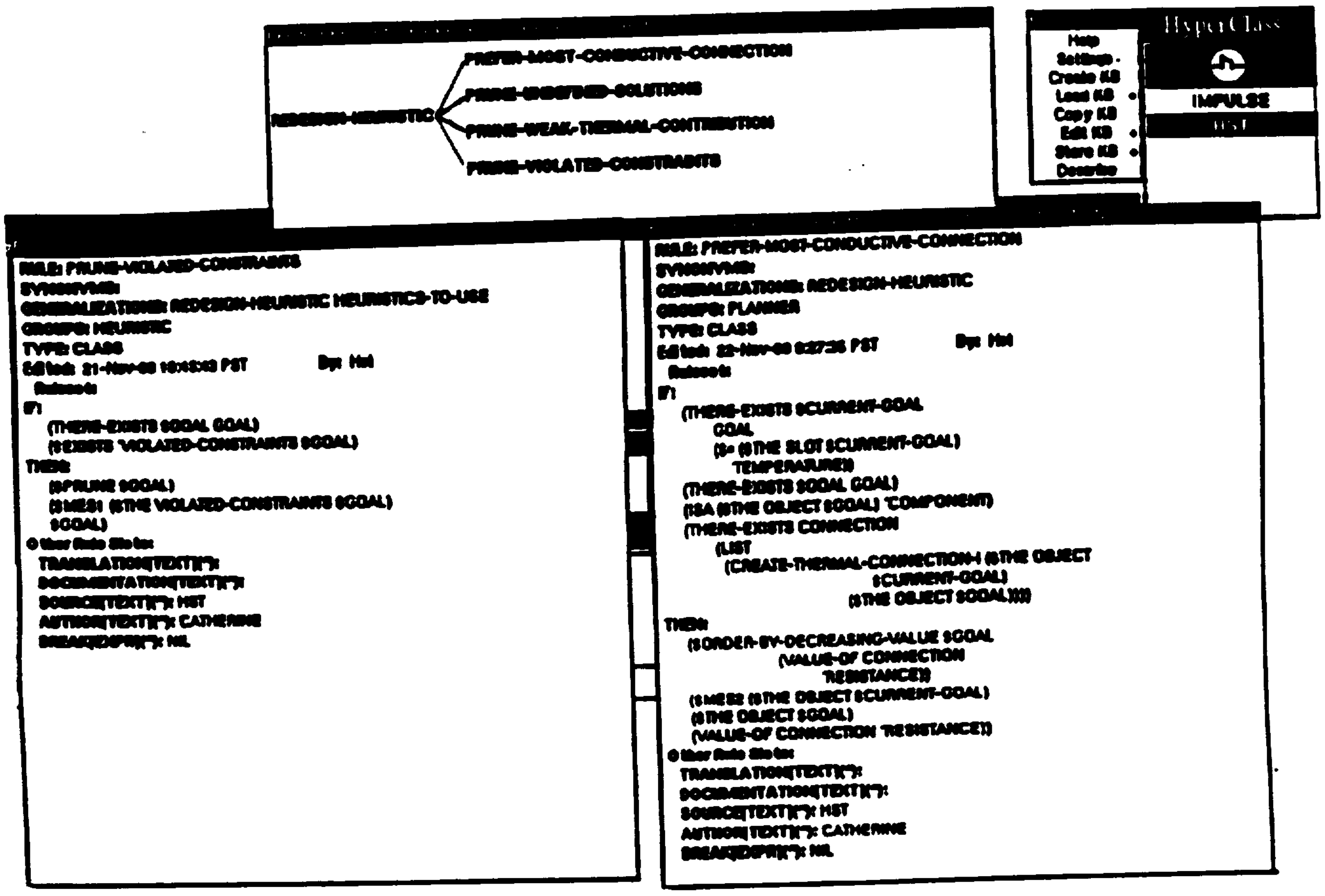

decrease BearingTemp
decrease RCEContribution
increase RCE-BearingResistance
increase CasingWallThickness
increase CasingThermalConstant
decrease RCETemp
decrease MotorCurrent
decrease MotorContribution
increase Motor-BearingResistance
increase ShaftLength
increase MotorThermalConstant
decrease MotorTemp
increase CoilRadius
decrease Motor-Current
decrease BearingFriction
increase BallRadius
increase BearingWidth
decrease MotorSpeed
decrease MotorCurrent
decrease TunnelContribution
decrease TunnelTemp
increase TunnelReflectivity
increase Tunnel-BearingResistance
increase CasingWallThickness
increase CasingThermalConstant
increase RCEBodyWidth
increase RCEThermalConstant
x → y : y is a redesign subgoal of x

Figure 10: Initial redesign goal tree

3.4. Goal pruning and ordering

The next step in the compilation process is to prune and order the nodes in the goal tree according to a set of redesign heuristics. This has the effect of transforming the set of *possible* redesign actions (represented by the leaves of the goal tree) into a smaller, prioritized sequence of *recommended* redesign actions. The end result of pruning and ordering the initial goal tree is the final redesign goal tree presented in Figure 8 above. The heuristics were developed based on conversations with the RWA designer. Each heuristic is represented in a separate frame as a rule that reasons about the goal tree structure. In this way, task-dependent heuristics can be isolated from the rest of the reasoning components and represented declaratively. This makes it easy to put in additional domain-specific redesign heuristics when new ones are acquired or when the system is applied to a different domain. The top window in Figure 11 shows the frame hierarchy of the redesign heuristics currently in the knowledge base.


REDESIGN-HEURISTIC
PREFER-MOST-CONDUCTIVE-CONNECTION
PRUNE-UNDEFINED-SOLUTIONS
PRUNE-WEAK-THERMAL-CONTRIBUTION
PRUNE-VIOLATED-CONSTRAINTS
Help
Settings
Create KB
Load KB
Copy KB
Edit KB
Store KB
Describe
HyperClass
IMPULSE
HST

```
RULE: PRUNE-VIOLATED-CONSTRAINTS
SYNONYMS:
GENERALIZATIONS: REDESIGN-HEURISTIC HEURISTICS-TO-USE
GROUPS: HEURISTIC
TYPE: CLASS
Edited: 21-Nov-88 [illegible] PST          By: Hst
  Subsets:
IF:
   (THERE-EXISTS $GOAL GOAL)
   ($EXISTS 'VIOLATED-CONSTRAINTS $GOAL)
THEN:
   ($PRUNE $GOAL)
   ($MES1 ($THE VIOLATED-CONSTRAINTS $GOAL)
    $GOAL)
Other Rule Slots:
  TRANSLATION[TEXT][*]:
  DOCUMENTATION[TEXT][*]:
  SOURCE[TEXT][*]: HST
  AUTHOR[TEXT][*]: CATHERINE
  BREAK[EXPR][*]: NIL
```

```
RULE: PREFER-MOST-CONDUCTIVE-CONNECTION
SYNONYMS:
GENERALIZATIONS: REDESIGN-HEURISTIC
GROUPS: PLANNER
TYPE: CLASS
Edited: 22-Nov-88 [illegible] PST          By: Hst
  Subsets:
IF:
   (THERE-EXISTS $CURRENT-GOAL
        GOAL
        ($= ($THE SLOT $CURRENT-GOAL)
           TEMPERATURE))
   (THERE-EXISTS $GOAL GOAL)
   (ISA ($THE OBJECT $GOAL) 'COMPONENT)
   (THERE-EXISTS CONNECTION
        (LIST
         (CREATE-THERMAL-CONNECTION-I ($THE OBJECT
                                     $CURRENT-GOAL)
                               ($THE OBJECT $GOAL))))
THEN:
   ($ORDER-BY-DECREASING-VALUE $GOAL
              (VALUE-OF CONNECTION
                    'RESISTANCE))
   ($MES2 ($THE OBJECT $CURRENT-GOAL)
   ($THE OBJECT $GOAL)
   (VALUE-OF CONNECTION 'RESISTANCE))
Other Rule Slots:
  TRANSLATION[TEXT][*]:
  DOCUMENTATION[TEXT][*]:
  SOURCE[TEXT][*]: HST
  AUTHOR[TEXT][*]: CATHERINE
  BREAK[EXPR][*]: NIL
```

Figure 11: Redesign heuristics

3.4.1. Pruning heuristics

The compiler currently uses three pruning heuristics. The first heuristic, *prune-violated-constraints*, (see the lower left window in Figure 11) eliminates any goals whose execution would violate pre-specified design constraints on quantities. For example, suppose one of the goals suggests decreasing the current to the motor. Further suppose that based on an analysis of qualitative equations relating motor current with motor torque, the effect of executing this action would be a decrease in motor torque. If either the motor current or the motor torque were constrained to be constant (e.g., to comply with specifications in a design document), the compiler would prune the suggested goal. (As discussed in Section 2.2.2.1, simple unary constraints on numeric quantities are represented in the Constraint slot of the corresponding quantity object.)

The second pruning heuristic, *prune-weak-thermal-connection*, is specific to situations involving redesign of thermodynamic properties. The basic idea is to prune any redesign actions involving components that are thermally insulated from the target component undergoing thermal redesign. For example, suppose the goal is to decrease the temperature of the RCE-bearing, and one of the recommended actions (i.e. leaf subgoals) is to change the width of the exterior bay door. If the thermal resistance between the RCE-bearing and the bay door is over a preset threshold, then the recommended action will be pruned. In other words, due to high thermal resistance, the recommended action on the bay door is expected to have a negligible thermal effect on the RCE-bearing, and should be omitted. The actual procedure for computing thermal resistance is described later in Section 4.3. This heuristic is a specific version of a more general, domain-independent heuristic that recommends avoiding modification of quantities that are only "weakly determinant" of the goal quantity, where "weakly determinant" is a domain-dependent notion.

The third pruning heuristic, ***prune-undefined-solution***, eliminates any subgoals for which the effects of their execution on the goal parameter cannot be determined from the given information. Such a situation arises due to the qualitative nature of the equations we used to infer the influence of a parameter upon another. For example, suppose we are given four parameters, w, x, y, and z with the following relations among them: An increase in either x or y causes an increase in w, and an increase in z causes an increase in x but a decrease in y. Then, if increasing w is our goal, we cannot decide whether increasing z will have a negative or positive effect on w.

3.4.2. Ordering heuristic

The ordering heuristic, ***prefer-most-conductive-connection*** (see the lower right window in Figure 11), is related to the second pruning heuristic. The thermal redesign actions that remain after applying the resistance threshold are ordered by value of increasing thermal resistance. A more general statement of the heuristic is to execute redesign actions so that quantities "strongly determinant" of the goal quantity are modified before "weakly determinant" quantities.

3.5. Redesign plan synthesis

The final compilation step involves synthesizing an abstract redesign plan that caches the recommended sequence of redesign actions for accomplishing the specified redesign goal. The plan is a type of macro-rule formed from the root and leaves of the final redesign goal tree. In particular, the root of the tree forms the rule's antecedent (the plan applicability conditions) and the ordered leaves of the tree form its consequent (the abstract plan, itself).[5] The final rule produced from the goal tree in Figure 8 is given at the beginning of Section 3.

3.6. Summary: Compiling redesign plans

Figure 12 summarizes the entire process of compiling redesign plans, starting with the general-purpose RWA device model and ending with a special-purpose redesign heuristic. Note how the information content of the original structure/behavior model is reduced by each successive compilation step. Furthermore, each successive model is more specially tuned and efficient for the requirements of the redesign task at hand. The compiler takes a number of different types of knowledge as input, ranging from problem specific information (e.g., the redesign goal, controllability assumptions) to problem class specific information (e.g., thermal redesign heuristics, system scoping assumptions). If any of these inputs change, the derived redesign plans should be recompiled.

[5]In practice, an antecedent containing only the specified redesign goal will be overly general and, as a result, may recommend the consequent redesign plan when it does not correctly apply. To remedy this situation, additional assumptions used in formulating the redesign plan must be included in the antecedent to restrict the consequent's domain of applicability. For example, it is possible to specify the scope of the redesign analysis, and the heuristics used to prune and order the redesign actions as part of the antecedent.

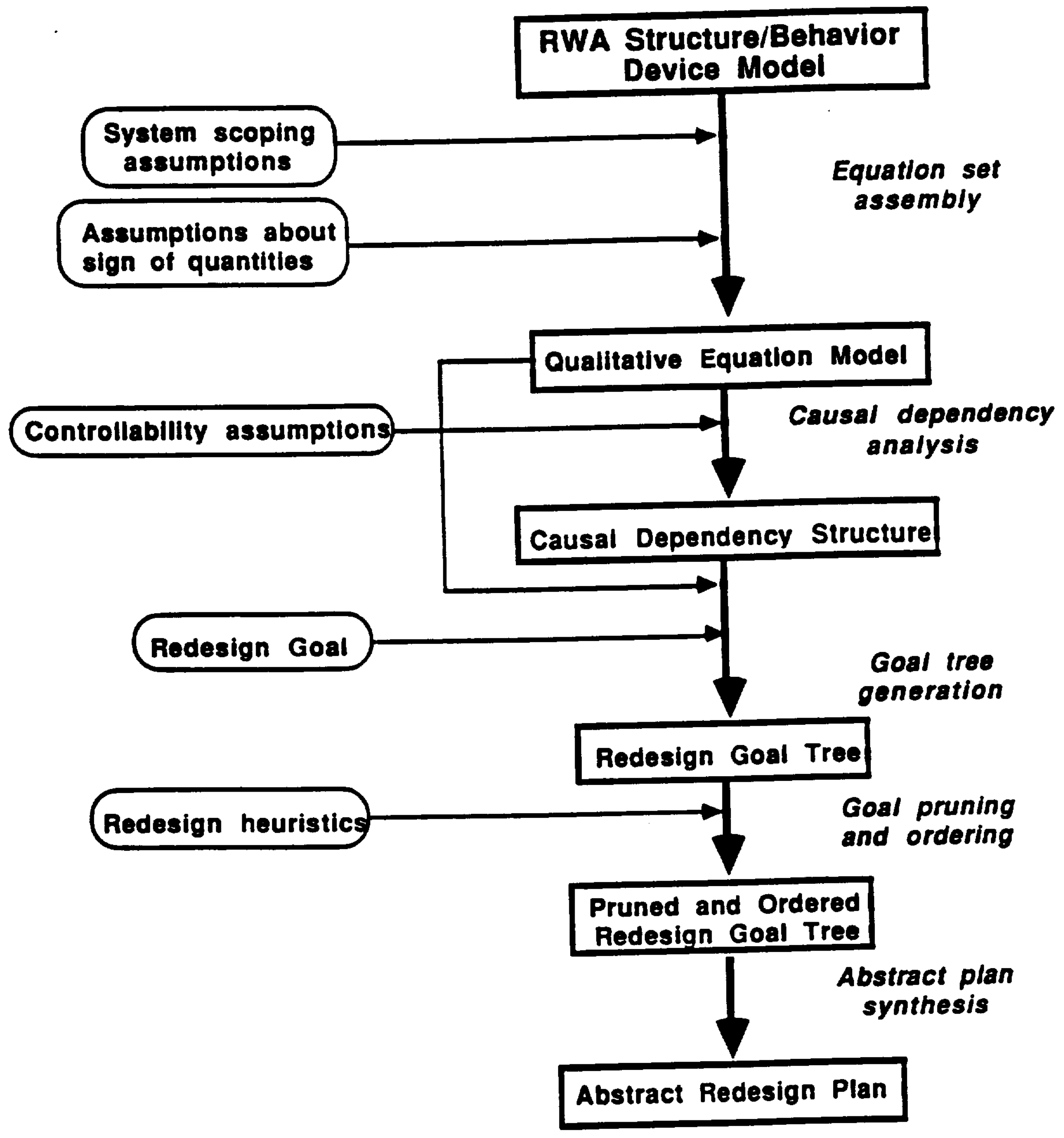

RWA Structure/Behavior Device Model
System scoping assumptions
Assumptions about sign of quantities
Equation set assembly
Qualitative Equation Model
Controllability assumptions
Causal dependency analysis
Causal Dependency Structure
Redesign Goal
Goal tree generation
Redesign Goal Tree
Redesign heuristics
Goal pruning and ordering
Pruned and Ordered Redesign Goal Tree
Abstract plan synthesis
Abstract Redesign Plan

Figure 12: Compilation of redesign plans

4. COMPILING DIAGNOSTIC RULES

The second knowledge compiler takes as input the same structure/behavior device model as the redesign plan compiler, but it produces a set of special-purpose "fault localization" rules for diagnosis, instead. The diagnostic rule compilation process differs markedly from the process described in Section 3, by which an abstract redesign plan is compiled. While the latter resembles macro-formation, the former involves partial evaluation of a very general fault localization rule. In describing the behavior of the compiler, we present *in reverse order* the sequence of three compilation steps being executed, starting with the "target" of the compilation activity -- a specific fault localization rule -- and working backwards toward its decompiled "source" – the general-purpose device model. This order of presentation reflects the "reverse engineering" methodology used in constructing the compiler.

4.1. Compiling Fault Localization Rules

Following is an example of a fault localization rule that the diagnostic compiler can produce:[6]

R2: *If* *Temperature of RCE-BEARING-SENSOR-22 is High*
and *Temperature of RCE-SENSOR-34 is OK*
and *Temperature of TUNNEL-SENSOR-101 is OK*
then *set Malfunction of RCE-BEARING-6 to True.*

To understand this rule, refer to Figure 13. The rule says that if the sensor for the RCE-bearing is abnormally high, and nearby sensor readings are normal, then there must be a malfunction within the RCE-bearing. On first analysis this rule appears incomplete because it only checks the sensor readings for the RCE and the tunnel, and omits other nearby components that could potentially influence the reading on the RCE-bearing sensor. For example, the motor generates considerable heat, and so do the PCE and the bay door (which heats up due to the sun). Why aren't these heat sources checked in the compiled rule? The answer is that the experts consider the influence of these heat sources to be negligible.

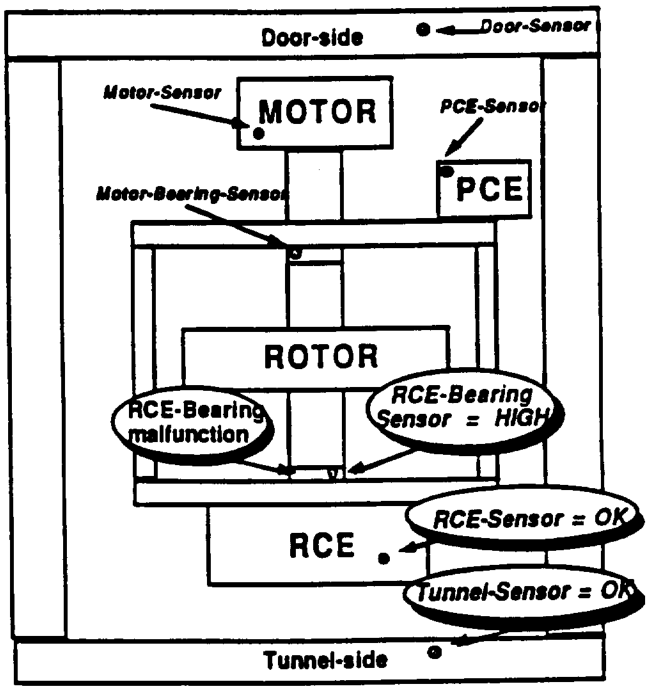


Figure 13: RWA component features referenced in rule R2

To produce R2, the rule compiler makes use of a simple, but general diagnostic fault localization model. Suppose $\{SRC_1, SRC_2, ..., SRC_n\}$ is a set of source components that produce some substance S (e.g., thermal energy), and suppose $\{SEN_1, SEN_2, ..., SEN_n\}$ is a set of corresponding sensors that measure the amount of S at each source component. Here is a general diagnostic rule that captures the fault localization idea:

[6]This rule was extracted from an actual "shallow" RWA diagnostic expert system built by Lockheed [Austin & Laffey 86].

```
R1:   If     Reading of SEN_i is Abnormal
      and    (forall k≠i | influences(SRC_k, SEN_i))
                    Reading of SEN_k is Normal
      then   set Malfunction of SRC_i to True.
```

Notice how the predicate *influences* captures the notion that only certain sources are capable of influencing the reading of a given sensor. To automatically compile rule R2 from rule R1, the compiler uses a technique called partial evaluation [Kahn 84]. Partial evaluation produces a specialized version of R1 by incorporating domain knowledge about the source components (SRCs) and the sensors (SENs). For example, if we are trying to localize faults within RCE-BEARING-6, we know the variable SEN_i in rule R1 must be bound to the sensor RCE-BEARING-SENSOR-22. This is due to our knowledge about the RWA device, which tells us which sensors are designed to measure which components. Furthermore, we know that RCE-BEARING-SENSOR-22 measures temperature, that its defined abnormal value is "high", and that its normal value is "ok". Again, this is specific domain knowledge pertaining to the particular sensor. (A different sensor might measure current, for example, and its abnormal reading might be "low", "13.6", or any other sensor-specific notion of abnormal.) By "folding" this knowledge into R1, we get the following rule:

```
R1.5:  If     Temperature of RCE-BEARING-SENSOR-22 is High
       and    (forall k≠i | influences(SRC_k, RCE-BEARING-SENSOR-22))
                     Reading of SEN_k is OK
       then   set Malfunction of RCE-BEARING-6 to True.
```

The final step from R1.5 to R2 can be achieved if we know the identity of all heat sources in the RWA, and know whether each heat source is capable of influencing RCE-BEARING-SENSOR-22 or not. For example, suppose we have available the simple model of thermal influences on the RWA sensors, as depicted in Figure 14.[7] In this model, a heat source is connected to a heat sensor if the source can influence the sensor's reading. Using this model, the rule compiler can evaluate the *forall* clause in R1.5, and correspondingly generates a separate clause in R2 for each of the two heat sources that influence RCE-bearing-sensor-22: RCE-201 and Tunnel-door-14.

Note that both R1 and R2 achieve the exact same result when fired on RCE-BEARING-6 – they both set its Malfunction slot to True. The fundamental difference between the two rules is that R2 trades off R1's generality for increased computational efficiency. R2 is less general than R1 because it only works for a single sensor: RCE-bearing-sensor-22. But R2 is more efficient than R1 because less computation is necessary in order to match R2's "if" part. Computing R1's "if" part involves matching all known heat sources and computing the "influences" predicate for each. If we have available a model such as the one in Figure 14, the amount of computational effort necessary to compute "influences" is minimal. In general, however, this type of pre-computed thermal influence model may not be available. In this case, it will be necessary to spend additional computational effort to compute the "influences" relations from more basic knowledge about the thermal structure of the RWA. This process is described in the following section.

[7]To simplify the presentation of the thermal sensor influence model, Figure 9 only displays influences for two of the RWA sensors.

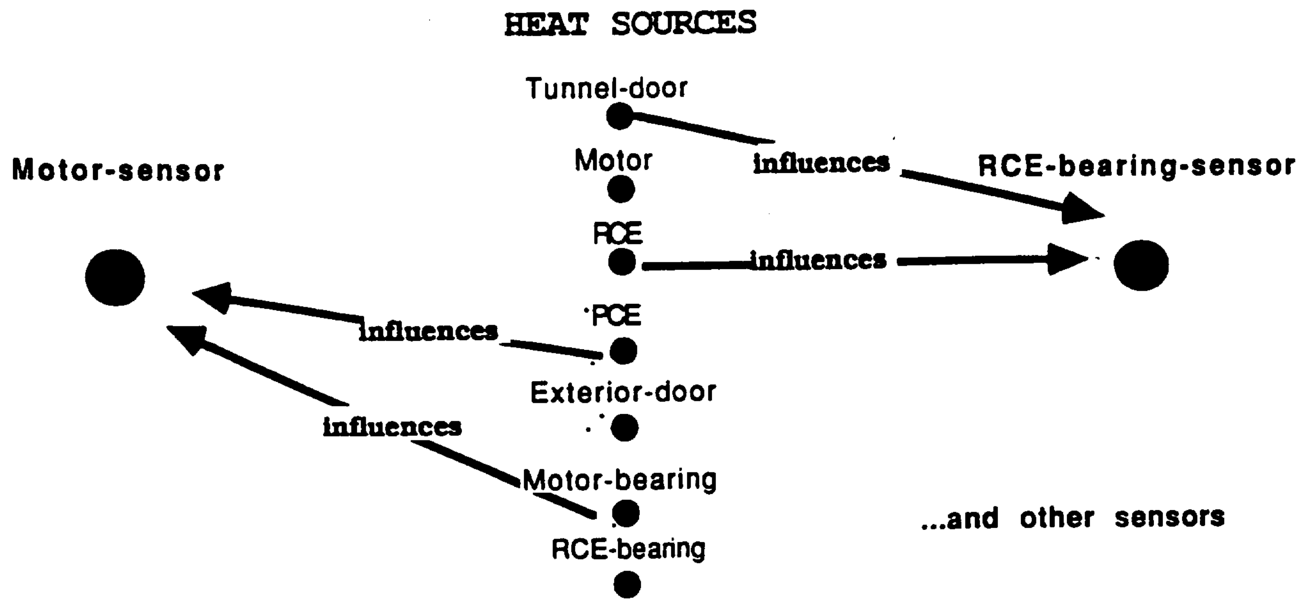


Figure 14: Thermal influence model for RWA

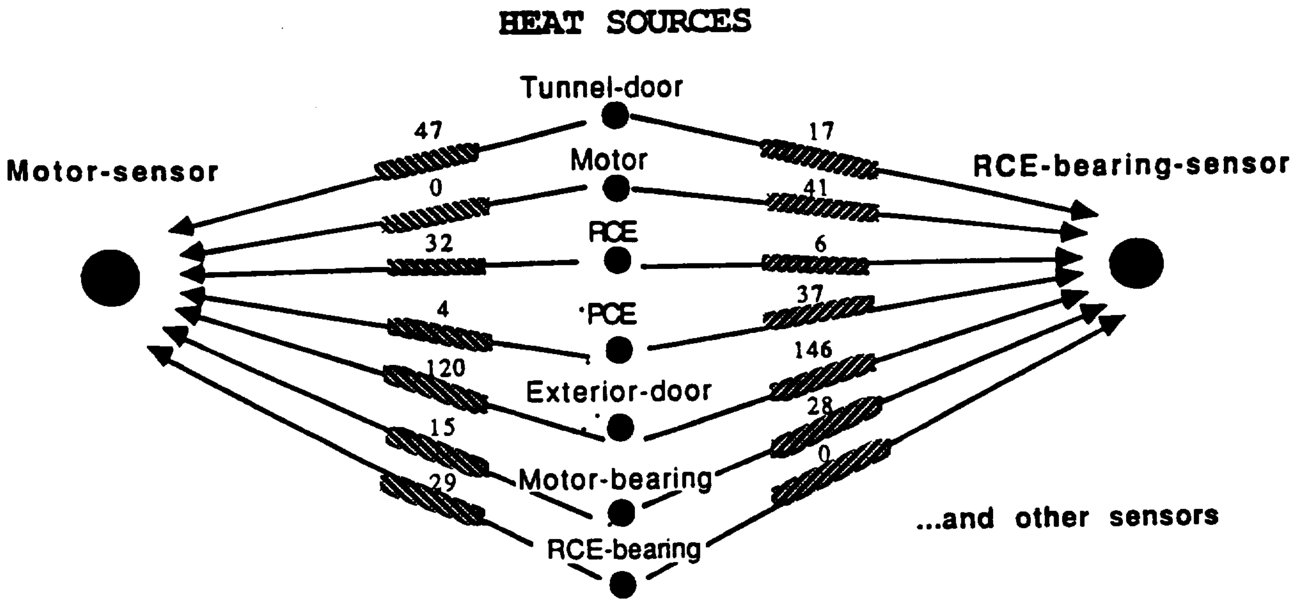


Figure 15: Thermal resistance model for RWA

4.2. Compiling the Thermal Influence Model

Suppose we have available a more sophisticated thermodynamic model of the RWA based on the notion of *thermal resistance*. In this model (Figure 15), there is a numeric thermal resistance value associated with each heat flow path linking a heat source and a heat sensor. The higher the resistance value, the harder it is for heat to flow along the path. It follows that the amount of thermal influence imposed by a given heat source on a given heat sensor is somehow inversely proportional to the amount of thermal resistance along the path between the source and sensor. One way of deriving the binary thermal influence model from a scalar thermal resistance model is to threshold the thermal resistance value. Any heat flow path with a resistance below the preset threshold will be considered a path of thermal influence when generating the thermal influence model. For example, with the threshold set at 20, the thermal resistance model in Figure 15 collapses into the thermal influence model in Figure 14.

Note that the transformation from resistance model to influence model is an information-losing transformation. In particular, the influence model incorporates several implicit assumptions that are made explicit in the resistance model. For example, the assumptions made in designating a heat source "influential" can be made explicit in the resistance model by specifying a threshold and a resistance metric. The relationship between the two thermal models is similar to the relationship between the diagnostic rules R1 and R2 above; there is a trade-off between generality and efficiency in these two models. It is more efficient to determine thermal influence by thresholding the influence model, but the model is not suited for much else. On the other hand, computing thermal influence with the resistance model is still possible, (although less efficient), and because the resistance model carries more information, it is more versatile. In particular, the resistance model can be used to answer a larger set of questions about thermal properties of RWA -- including, but not strictly limited to questions about thermal influence.

4.3. Compiling the Thermal Resistance Model

We can carry this trade-off between generality and efficiency once step further by decompiling the thermal resistance model in terms of the RWA device model specified in Section 2.2. Unlike the thermal resistance model, which provides a special-purpose model of certain thermodynamic properties of the RWA, the device model is more general-purpose, and can be used to answer a wider variety of questions about the device. Using information about physical structure and material composition made available by this RWA model, plus knowledge about thermodynamics and heat-flow, the diagnostic rule compiler can generate the special-purpose thermal resistance model in Figure 15.

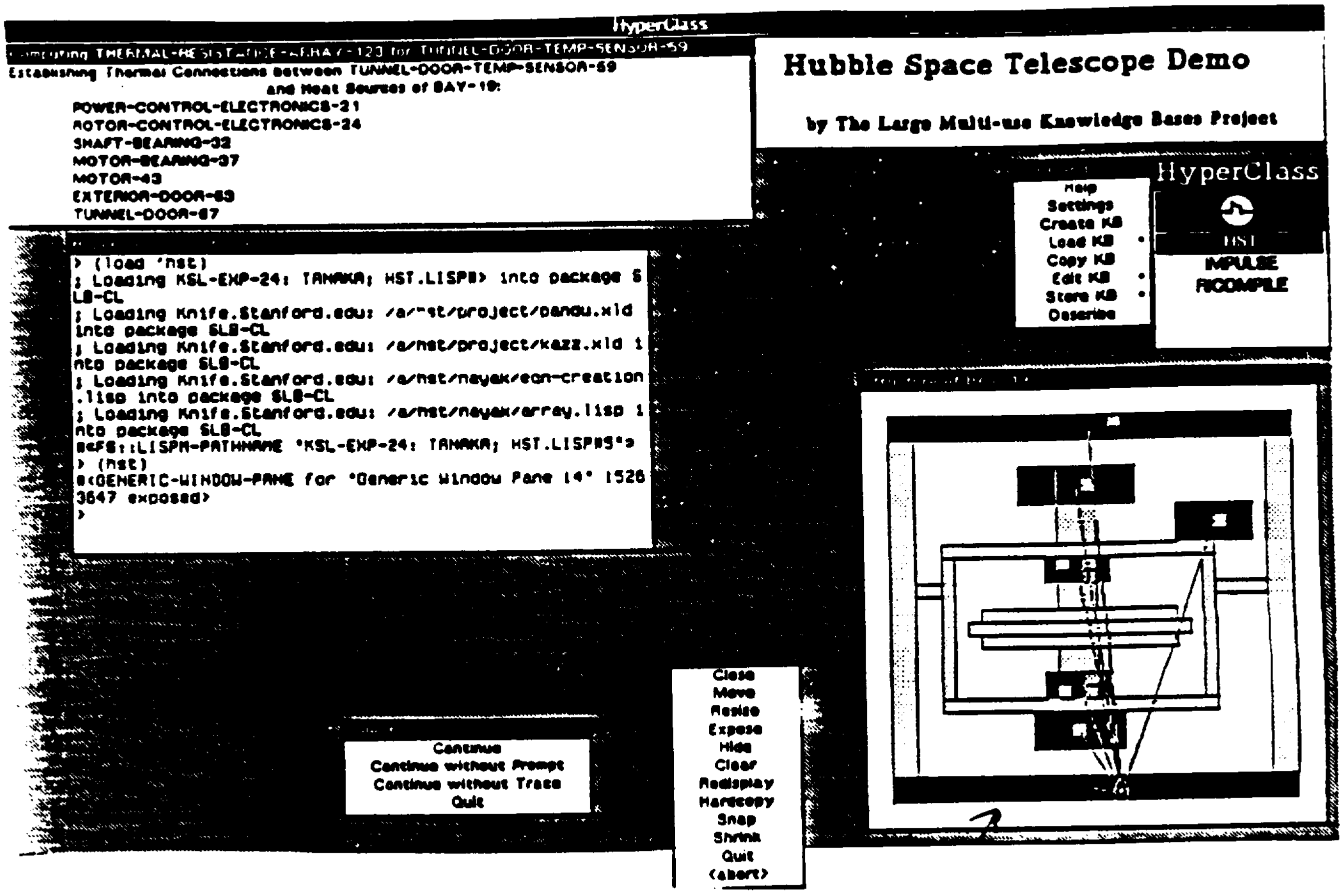
The generation of a thermal resistance model for the RWA is accomplished in two steps: 1) establishment of heat flow paths and 2) computation of thermal resistance along those paths. First, the compiler establishes a thermal connection between each heat source and heat sensor in the RWA (Figure 16). Thermal connections are represented in a manner similar to physical connections, as discussed in Section 2.2.1.3 above. For example, the following thermal connection object represents the heat flow path between RCE-BEARING-SENSOR-22 and TUNNEL-DOOR-43:

*Object: RCE-BEARING*TUNNEL-DOOR-128*
Type: individual
Generalizations: THERMAL-CONNECTION
Specializations: nil
Comp1: RCE-BEARING-SENSOR-22
Comp2: TUNNEL-DOOR-43
*Resistance: RCE-BEARING*TUNNEL-DOOR-RESISTANCE-221*

The filler of the Resistance slot is a QUANTITY instance, which records the actual numeric value of the thermal resistance. The compiler knows whether a given component within the RWA generates heat because this information is explicitly recorded in one of the component's slots.[8]

The actual thermal resistance between two components is computed by one of two equations associated with the class of THERMAL-RESISTANCE quantities. The first equation uses a radiative model of heat flow to compute the thermal resistance, whereas the second equation uses a conductive model.

Object: THERMAL-RESISTANCE
Type: class
Generalizations: QUANTITY
Specializations: nil
Possible-equations: (Radiative-resistance-eqn*s,Conductive-resistance-eqn*s)
*Radiative-resistance-eqn*s:* ; filled when instantiated
*Conductive-resistance-eqn*s:* ; filled when instantiated


```
Computing THERMAL-RESISTANCE-ARRAY-123 for TUNNEL-DOOR-TEMP-SENSOR-59
Establishing Thermal Connections between TUNNEL-DOOR-TEMP-SENSOR-59
                    and Heat Sources of BAY-19:
    POWER-CONTROL-ELECTRONICS-21
    ROTOR-CONTROL-ELECTRONICS-24
    SHAFT-BEARING-32
    MOTOR-BEARING-37
    MOTOR-43
    EXTERIOR-DOOR-63
    TUNNEL-DOOR-67
```

```
> (load 'hst)
; Loading KSL-EXP-24: TANAKA; HST.LISP#> into package S
LB-CL
; Loading Knife.Stanford.edu: /a/~st/project/pandu.xld
into package SLB-CL
; Loading Knife.Stanford.edu: /a/hst/project/kazz.xld i
nto package SLB-CL
; Loading Knife.Stanford.edu: /a/hst/nayak/eqn-creation
.lisp into package SLB-CL
; Loading Knife.Stanford.edu: /a/hst/nayak/array.lisp i
nto package SLB-CL
#<FS::LISPM-PATHNAME "KSL-EXP-24: TANAKA; HST.LISP#S">
> (hst)
#<GENERIC-WINDOW-PANE for "Generic Window Pane 14" 1526
3647 exposed>
>
```

Figure 16: Establishing thermal connections

[8]In a more complete model of the RWA device, various physical processes – such as heat generation, heat flow, and energy conversion – might be represented explicitly (e.g., as in [Forbus 84]). In this case, the system might be able to automatically infer which components generate heat using more basic principles.

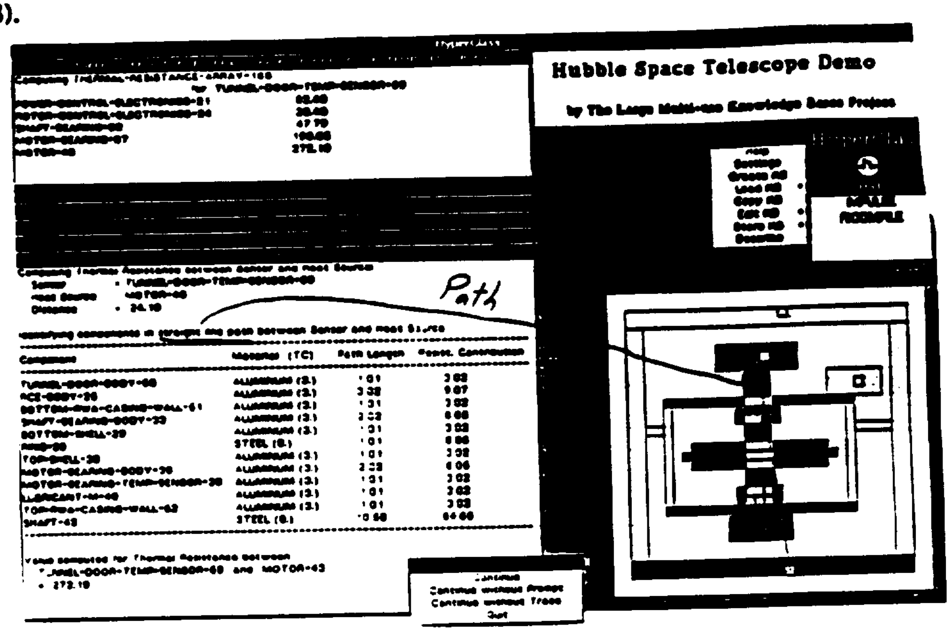
The radiative equation consists of a procedure call[9] which computes the resistance of a thermal connection as follows: Using the bounding box spatial representation (Section 2.2.1.2), a straight line path is computed between the centers of two thermally-connected components (i.e., between the centers of the heat source and the heat sensor). Then the resistance is calculated according to the following formula:

$$\textit{Thermal Resistance}_{rad} = d_{ij} * \frac{\Sigma_{k=1,n}\,(\textit{thermal-constant}_k * \textit{path-length}_k)}{n}$$

where: d_{ij} is the linear distance between the center of components i and j
k is an index over the components intersecting the straight line path between components i and j
n is the number of intersecting components
thermal-constant$_k$ refers to the thermal constant associated with component k's material
path-length$_k$ refers to the length of intersection between component k and the straight line path

The radiative resistance is a weighted average of the thermal constants associated with the materials of each primitive component in the straight line path between the heat source and sensor. The weighting is proportional to the thickness of the components in the heat flow path. This is illustrated by the display in Figure 17. Although the formula is not accurate from a thermodynamic perspective, it provides a heuristic estimate of the radiative resistance.

Conductive resistance is computed by a similar procedure. However, instead of using a straight line path between the heat source and sensor, the conductive procedure searches for a connected physical path between the two components using the RWA device model representation of physical connections (see Section 2.2.1.3).


Hubble Space Telescope Demo
by The Large Multi-use Knowledge Bases Project
Path

```
Computing THERMAL-RESISTANCE-ARRAY-188
                          for TUNNEL-DOOR-TEMP-SENSOR-69
POWER-CONTROL-ELECTRONICS-21              62.40
MOTOR-CONTROL-ELECTRONICS-24              30.40
SHAFT-BEARING-08                          47.79
MOTOR-BEARING-07                         190.65
MOTOR-43                                 272.10

Computing Thermal Resistance between Sensor and Heat Source
   Sensor       = TUNNEL-DOOR-TEMP-SENSOR-69
   Heat Source  = MOTOR-43
   Distance     = 24.10

Identifying components in straight line path between Sensor and Heat Source
---------------------------------------------------------------------------
Component                          Material (TC)   Path Length  Perc. Contribution
---------------------------------------------------------------------------
TUNNEL-DOOR-BODY-68                ALUMINUM (3.1)     1.01          3.02
ACE-BODY-36                        ALUMINUM (3.1)     3.38          9.07
BOTTOM-RWA-CASING-WALL-61          ALUMINUM (3.1)     1.31          3.02
SHAFT-BEARING-BODY-33              ALUMINUM (3.1)     3.02          6.06
BOTTOM-SHELL-29                    ALUMINUM (3.1)     1.01          3.02
RING-30                            STEEL (6.)         1.01          6.06
TOP-SHELL-38                       ALUMINUM (3.1)     1.01          3.02
MOTOR-BEARING-BODY-36              ALUMINUM (3.1)     2.02          6.06
MOTOR-BEARING-TEMP-SENSOR-38       ALUMINUM (3.1)     1.01          3.02
LUBRICANT-M-40                     ALUMINUM (3.1)     1.01          3.02
TOP-RWA-CASING-WALL-62             ALUMINUM (3.1)     1.01          3.02
SHAFT-43                           STEEL (6.)        10.80         64.06
---------------------------------------------------------------------------

Value computed for Thermal Resistance between
  TUNNEL-DOOR-TEMP-SENSOR-69 and MOTOR-43
  = 272.10
```

```
Continue
Continue without Prompt
Continue without Trace
Quit
```

Figure 17: Computing radiative resistance

[9]The equation is too complex to state in the system's current equation language, which allows only simple algebraic constraints between quantities. See Section 2.2.2.2.

4.4. Summary: Diagnostic Rule Compilation

Figure 18 summarizes the entire rule compilation process. In the first step, the RWA structure/behavior device model is compiled into a thermal resistance model by applying one of two heat flow equations. Next, the thermal resistance model is reduced to a thermal influence model by applying a thresholding procedure. Finally, a set of RWA-specific fault localization rules is produced by partially evaluating a generalized fault localization rule based on information in the thermal influence model. At each step in this process, the information content of the resulting model is reduced, yielding a more efficient, but less general-purpose model of the RWA.

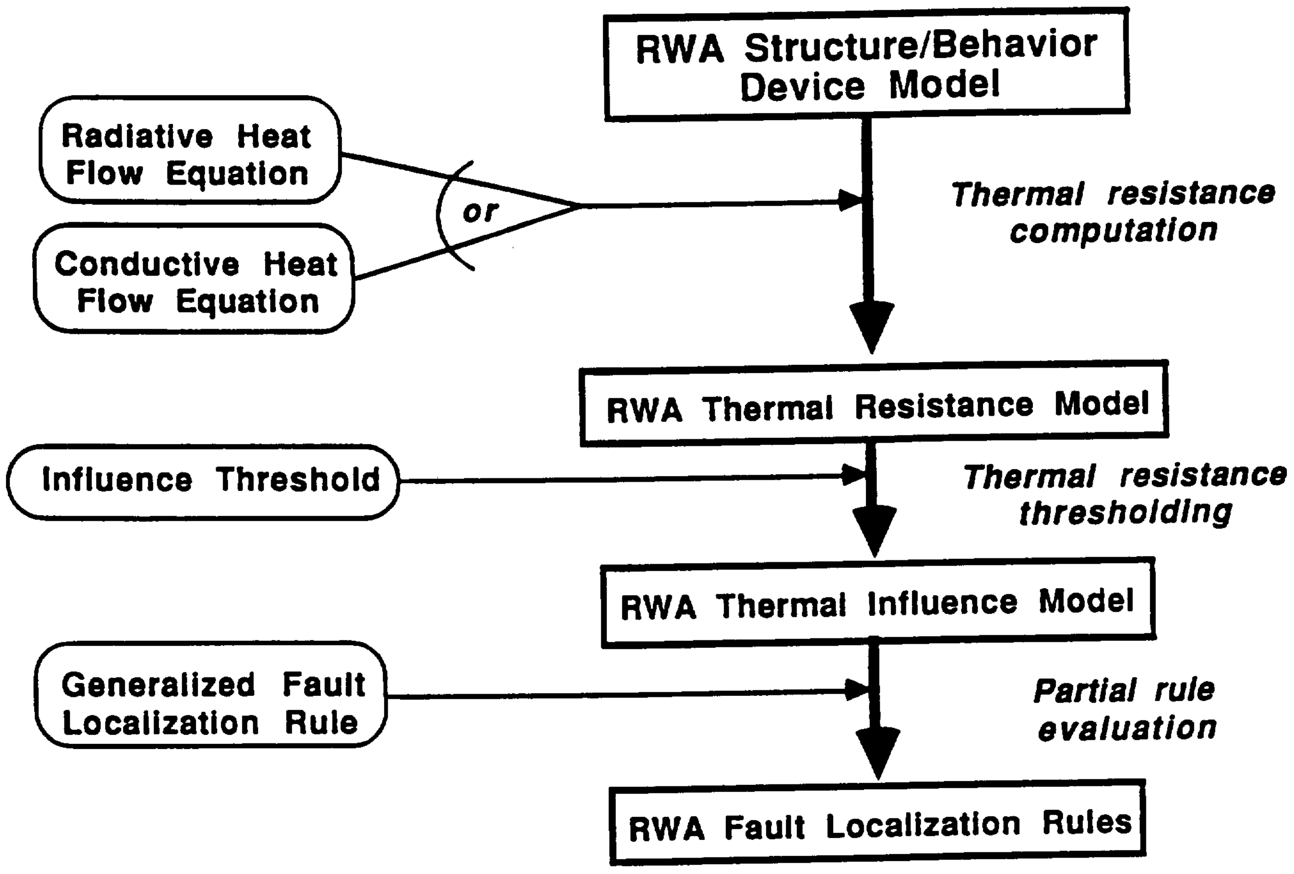

RWA Structure/Behavior Device Model
Radiative Heat Flow Equation
or
Conductive Heat Flow Equation
Thermal resistance computation
RWA Thermal Resistance Model
Influence Threshold
Thermal resistance thresholding
RWA Thermal Influence Model
Generalized Fault Localization Rule
Partial rule evaluation
RWA Fault Localization Rules

Figure 18: Compilation of diagnosis rule

5. DISCUSSION

The examples described above illustrate how the rule compilation approach can be applied to derive two types of shallow knowledge (abstract redesign plans and fault localization rules) from a common underlying device model. Figures 12 and 18 summarize both the steps involved and the intermediate models derived as by-products during the design and diagnostic compilation processes, respectfully. Note that although the intermediate models are derived by either the redesign or the diagnostic compiler, these models are not necessarily task specific. For example, the qualitative equation, thermal resistance, and causal dependency models (in Figure 12), as well as the thermal resistance and thermal influence models (in Figure 18) are quite general-purpose. These models could potentially be useful in a number of different application tasks. In fact, the thermal influence model is actually used in both our redesign task (where it is used to prune from consideration any redesign activities focused on thermally insulated components) and our diagnosis task (where it is used to determine which heat sources could cause an abnormally high thermal reading for a given heat sensor). Toward the end of the compilation process, the derived models take on an increasingly task specific flavor with the incorporation of more task specific knowledge. For example, the redesign compiler uses redesign goals and heuristics to generate goal trees that are specific to the redesign task.

The final end-product of the compilation process is a set of highly tuned, task specific rules or plans. However, in contrast with traditional expert system rules, compiled rules/plans are more robust under conditions of change over time. In particular, when underlying deep models change, the rules or plans can be recompiled to reflect the updated information. For example, suppose that after extensive testing, NASA designers decide to choose a lighter but more conductive material for the motor shaft. In this case, the amount of heat transferred from the motor to the RCE bearing via the motor shaft would increase. Correspondingly, the priority of redesign actions associated with decreasing motor heat should increase. The diagnostic and redesign compilers facilitate the process of automatically updating the now-invalid[10] abstract redesign plan (see beginning of Section 3). All the user needs to do is change the shaft material in the RWA device model (Section 2.2), and re-run the compilers. Resulting changes in the thermal resistance and thermal influence model will interact with the redesign ordering heuristic to cause a re-ordering of the leaves in the goal tree and, correspondingly, in the actions in the final redesign plan.

Aside from making systems sensitive to changes in underlying models, the rule compilation approach has the potential to enhance system robustness in a second sense. In particular, if any of the shallow rules are found to be incorrect, the system should be able to (1) justify the suspect rules in terms of the underlying models, then (2) identify the potentially incorrect assumptions or approximations used in producing the faulty shallow rule, and finally (3) fix the rule by correcting assumptions and recompiling a new rule. An approach to the rule justification process that is consistent with our viewpoint has been pursued by Swartout in his work on the XPLAIN system [Swartout 83]. Swartout reasons that to provide proper justification of expert system behavior, a system must have access to underlying domain models and to goals of the expert system's designer. XPLAIN contains an automatic programming component capable of generating a portion of an expert system based on this type of knowledge. Justifications are phrased in terms of the decisions made by the automatic expert system derivation. Similarly, justifications for the RWA rules can be phrased in terms of the model transformation decisions made by the appropriate knowledge compiler.

After a system succeeds in reconstructing the justification underlying a faulty shallow rule, the next step would be to identify and fix incorrect assumptions or approximations used in producing that rule. This identification process is essentially a blame assignment process. In order to isolate incorrect assumptions,

[10] Assuming the plan compiler explicitly records the chain of models and assumptions used in deriving the plan, it is straightforward to determine whether a given change in an underlying model will impact the final plan's validity. A mechanism similar in spirit to those used by reason maintenance systems [Doyle 79] is required.)

the system requires knowledge about which assumptions are more likely to be faulty. For example, some of the assumptions compiled into abovementioned rule R2 include assumptions about RWA component attributes like position and material, as well as the value used for thermal resistance thresholding. The RWA component attributes are less likely to be incorrect than the threshold value because they have been obtained from the design specifications and reflect the actual device. On the other hand, the threshold setting is a heuristic value based on little direct empirical evidence and lacking solid theoretical underpinnings. To aid in this type of blame assignment process, Smith describes a language for describing assumptions underlying rules [Smith et al.]. The language allows the user to specify different assumption types (e.g., default, definitional, theoretical, statistical) and to specify dependencies among assumptions (e.g., abductive, deductive) that are used in propagating information about assumptions.

Although the overall rule compilation approach is promising, the work reported in this paper takes only the first steps toward evaluating the feasibility of this approach. We have illustrated two examples of rule compilation, but we have not developed a general architecture or mechanism for accomplishing knowledge compilation. Although the device representation is general-purpose and the rule compilation procedure will work for arbitrary devices described in this representation, the compilation steps implemented by the two knowledge compilers are "hard-wired" for the type of diagnosis and redesign problems illustrated in this paper. The construction of a general-purpose rule compiler raises a number of very difficult research questions. To understand the general issues, it is fruitful to view rule compilation as a search process through a space of models. The starting node is the general-purpose device model, the operators are approximating model transformations (e.g., thresholding, macro-formation), and the terminal node represents an efficient task-specific model. This view is generally consistent with the perspective taken in much previous work on knowledge compilation [Mostow 81, Keller 87, Ellman 88]. Some of the basic questions that must be answered before we can understand whether it is possible to build completely automated rule compilers include:

- How does one design a language for declaratively describing models, assumptions, and transformations on models?

- What set of model transformation operators is sufficient to handle compilation for a given type of task? Is there a significant overlap in the model transformation operators necessary to compile rules for different types of tasks?

- Is it possible to provide the search control necessary to select from the set of all available model transformation operators to apply at any given point in the search?

- Is it possible to recognize a terminal state in the model transformation search? I.e., what constitutes an "efficient task-specific model"?

Note that the rule compilation approach could be useful even if total automation of the process is not feasible. For example, a knowledge engineer might interactively provide the search control necessary to select and apply the model transformation operators. In terms of enhancing robustness, the important point is that the various device models and assumptions be made explicit, along with the transformations that bridge the gap between models at different levels of detail. This way, the (human-guided) compilation process is recorded, and recompilation can proceed automatically when the underlying model changes. In addition, the links back from the shallow model to the underlying models are preserved, and can be traversed when assumptions are violated and it is necessary to "fall back" on underlying models.

We do not believe that every high performance expert systems can be constructed solely using the rule compilation methodology. Sometimes, expert system rules merely encode the expert's experience as simple empirical associations between patterns and actions, and there is no underlying theory to justify the associations. In these cases, it is difficult to imagine how a program could compile such rules. However, in many domains, an underlying theory is available to provide some degree of justification for the expert rules. In these cases, we believe the rule compilation methodology is appropriate. As most expert systems contain a mix of theory-based and empirical associations, we would expect rule compilation to be appropriate for some rules, but not all, in a given expert system.

ACKNOWLEDGMENTS

We would like to thank the other members of the LMKB project for their contribution to this effort. This research has been supported by NASA under Grant # NCC2-537. The research reported in this chapter was performed at Stanford Knolwedge Systems Laboratory during a period from 1987 to 1989.

APPENDIX A

Following is a list of the qualitative differential equations collected during the equation set assembly process described in Section 4.1. This set of equations constitutes a model of the qualitative behavior of the RWA. The equations are in a canonical form with zero on the right hand side of the equation. There is one equation of the form [Q'] = 0 for each exogenous quantity Q known to the system (see equations 1 - 11). The notation [Q'] stands for the sign (+, -, 0) of the time derivative of Q. See [Iwasaki 89] for a discussion of qualitative arithmetic.

1) [MotorCurrent'] = 0

2) [CasingWallThickness'] = 0

3) [CasingThermalConstant']= 0

4) [RCEBodyWidth'] = 0

5) [RCEThermalConstant'] = 0

6) [BearingWidth'] = 0

7) [ShaftLength'] = 0

8) [MotorThermalConstant'] = 0

9) [CoilRadius'] = 0

10) [CoilDensity'] = 0

11) [TunnelReflectivity'] = 0

12) - [RCEBearingResistance'] + [RCETemp'] - [RCEContribution'] = 0

13) [MotorCurrent'] - [RCETemp'] = 0

14) [CasingWallThickness'] + [CasingThermalConstant'] + [RCEBodyWidth'] + [RCEThermalConstant'] - [TunnelBearingResistance'] = 0

15) - [TunnelBearingResistance'] + [TunnelTemp'] - [TunnelContribution'] = 0

16) - [MotorBearingResistance'] + [MotorTemp'] - [MotorContribution'] = 0

17) - [BearingWidth'] + [BallRadius'] = 0

18) [BallRadius'] - [MotorSpeed'] + [BearingFriction'] = 0

19) [TunnelContribution'] + [RCEContribution'] + [MotorContribution'] + [BearingFriction'] - [BearingTemp'] = 0

20) [ShaftLength'] + [MotorThermalConstant'] - [MotorBearingResistance'] = 0

21) [MotorCurrent'] - [MotorTorque'] = 0

22) - [MotorCurrent'] + [MotorSpeed'] = 0

23) - [CoilRadius'] + [MotorCurrent'] - [MotorTemp'] = 0

24) [CoilDensity']+ [CoilRadius'] - [CoilWeight'] = 0

25) [CasingWallThickness'] + [CasingThermalConstant'] - [RCEBearingResistance'] = 0

26) [TunnelReflectivity ']+ [TunnelTemp'] = 0

REFERENCES

[Anderson 86] J.R. Anderson, "Knowledge Compilation: The general learning mechanism". In *Machine Learning, Volume II,* R.S. Michalski, J.G. Carbonell, and T.M. Mitchell (editors), Morgan Kaufmann, 1986.

[Araya & Mittal 87] A. Araya and S. Mittal, "Compiling Design Plans from Descriptions of Artifacts and Problem Solving Heuristics". In *Proceedings IJCAI-87,* pp. 552-558, August 1987.

[Austin & Laffey 86] A. Austin and T. Laffey, "Knowledge-based Analysis of Telemetry Data on the Hubble Space Telescope". Technical report, Lockheed AI Center, 1986.

[Brown & Sloan 87] D.C. Brown and W.N. Sloan, "Compilation of Design Knowledge for Routine Design Expert Systems: An initial view". In *Proceedings ASME International Computers in Engineering Conference,* New York, NY, Vol. 1, pp. 131-136, 1987.

[Chandrasekaran & Mittal 83] B. Chandrasekaran and S. Mittal, "Deep versus Compiled Knowledge Approaches to Diagnostic Problem-solving". *International Journal of Man Machine Studies,* Vol. 19, pp. 425-436, May 1983.

[Davis 84] R. Davis, "Diagnostic Reasoning Based on Structure and Behavior",*Artificial Intelligence,* vol. 24, nos. 1-3, pp. 347-410, 1984.

[de Kleer & Brown 84] J. de Kleer and J.S. Brown, "A Qualitative Physics Based on Confluences", *Artificial Intelligence,* vol. 24, no. 1-3, pp. 7-84, 1984.

[Dietterich 86] T.G. Dietterich (ed.), *Proceedings of the Workshop on Knowledge Compilation,* Inn at Otter Crest, Oregon, Oregon State University technical report, 1986.

[Doyle 79] J. Doyle, "A Truth Maintenance System". *Artificial Intelligence,* vol. 12, no. 3, pp. 231-272, 1979.

[Ellman 88] T. Ellman, "Approximate Theory Formation: An Explanation-Based Approach". In *Proceedings AAAI-88,* St. Paul, Minnesota, pp. 570-574, August 1988.

[Forbus 84] K.D. Forbus, "Qualitative Process Theory". *Artificial Intelligence,* vol. 24, nos. 1-3, 1984.

[Gelman et al. 88] A. Gelman, S. Altman, M. Palakoff, K. Doshi, C. Manago, T.C. Rindfleisch, and B.G. Buchanan, "FRM: An Intelligent Assistant for Financial Resource Management". In *Proceedings AAAI-88,* St. Paul, Minnesota, pp. 31-36, August 1988.

[Genesereth 84] M.R. Genesereth, "The Use of Design Descriptions in Automated Diagnosis",*Artificial Intelligence,* vol. 24, no. 1-3, pp. 411-436, 1984.

[Iwasaki & Simon 86] Y. Iwasaki and H. A. Simon, "Causality in Device Behavior", *Artificial Intelligence,* vol. 29, no. 1, pp. 3-32, 1986.

[Iwasaki 89] Y. Iwasaki, "Qualitative Physics", *The Handbook of Artificial Intelligence, Vol. 4,* A. Barr, P.R. Cohen, and E.A. Feigenbaum (eds.), Addison-Wesley, Reading, MA, 1990.

[Kahn 84] K.M. Kahn, "Partial Evaluation, Programming Methodology, and Artificial Intelligence", *AI Magazine,* vol. 5, no. 1, pp. 53-57, 1984.

[Keller 87] R.M. Keller, *The Role of Explicit Contextual Knowledge in Learning Concepts to Improve Performance*, Ph.D. thesis (technical report # ML-TR-7), Department of Computer Science, Rutgers University, 1987.

[Keller et al. 87] R.M. Keller, B.G. Buchanan, E.A. Feigenbaum, "Development of a Reusable Knowledge Base for Space Applications". In *Proceedings of the Second Annual NASA Ames Artificial Intelligence Research Forum*, pp. 357-365. NASA, Palo Alto, CA, Nov. 1987.

[Laird et al. 87] J.E. Laird, P.S. Rosenbloom, and A. Newell, "Soar: An architecture for general intelligence", *Artificial Intelligence*, vol. 33, no. 3, 1987.

[LMSC 84] *Support Systems Module System Procedure for Pointing and Control Subsystem (SE-23, Vol. V)*, Lockheed Missiles and Space Company document # D889545A, 1984.

[Mostow 81] D.J. Mostow, *Mechanical Transformation of Task Heuristics into Operational Procedures*, Ph.D. thesis (technical report # CMU-CS-81-113), Department of Computer Science, Carnegie-Mellon University, 1981.

[Perkins & Austin 87] W.A. Perkins and A. Austin, "Experiments with Temporal Reasoning Applied to Analysis of Telemetry Data". In *Space Station Automation III*, vol. 851, pp. 39-46, Society of Photo-Optical Instrumentation Engineers, 1987.

[Sembugamoorthy & Chandrasekaran 86] V. Sembugamoorthy, and B. Chandrasekaran, "Functional Representation of Devices and Compilation of Diagnostic Problem-Solving Systems". In Kolodner, J.L. and Riesbeck, C.K. (editors), *Experience, Memory, and Reasoning*, Lawrence Erlbaum Associates, Hillsdale, NJ, 1986.

[Smith 83] R.G. Smith, "Strobe: Support for Structured Object Knowledge Representation". In *Proceedings IJCAI-83*, Karlsruhe, Germany, pp. 122-129, August 1983.

[Smith et al. 85] R.G. Smith, H.A. Winston, T.M. Mitchell, and B.G. Buchanan, "Representation and Use of Explicit Justifications for Knowledge Base Refinement". In *Proceedings IJCAI-85*, Los Angeles, CA, pp. 673-680, August 1985.

[Swartout 83] W.R. Swartout, "XPLAIN: A System for Creating and Explaining Expert Consulting Programs", *Artificial Intelligence*, vol. 21, no. 3, pp. 285-325, 1983.